ARM® System Memory Management Unit Architecture Specification

SMMU architecture version 2.0

ARM®

ARM IHI 0062D.c (ID070116)

ARM System Memory Management Unit Architecture Specification
SMMU architecture version 2.0

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
23 March 2012	A	Confidential Beta	Issue A. Beta release for architecture version 1.0.
18 December 2012	B	Non-Confidential	Issue B. Final release for architecture version 1.0.
16 September 2013	C.a	Non-Confidential Beta	Issue C. Beta release for architecture version 2.0.
13 January 2015	D	Non-Confidential	Issue D full release for architecture version 2.0 with architecture extensions.
28 January 2015	D.a	Non-Confidential	Non-Confidential Issue D full release. Architecture version 2.0 with architecture extensions.
15 July 2015	D.b	Non-Confidential	Issue D.b update with corrections and clarifications.
30 June 2016	D.c	Non-Confidential	Issue D.c update with corrections and clarifications.

Proprietary Notice

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

Web Address

http://www.arm.com

Contents
ARM System Memory Management Unit Architecture Specification SMMU architecture version 2.0

Preface

This preface introduces the *ARM® System Memory Management Unit Architecture Specification*. It contains the following sections:

About this specification

This specification introduces the ARM System MMU (SMMU) architecture.

Intended audience

This specification is written for readers who are familiar with system memory management concepts, but who do not necessarily have any experience of the ARM architecture.

Using this specification

The information in this specification is organized into the following chapters:

Chapter 1 *Introduction*

Read this for an introduction to the SMMU architecture.

Chapter 2 *SMMU Operation*

Read this for information about the steps that the SMMU performs on receiving a memory access request.

Chapter 3 *The Fault Model*

Read this for information about fault conditions the SMMU might encounter, and how the SMMU handles these faults.

Chapter 4 *Address Translation Operations*

Read this for information about address translation operations that are initiated using software-accessible registers.

Chapter 5 *Coherency Issues and Cache Maintenance Operations*

Read this for information about the relationship between controlling software and the SMMU, including optional support for coherent translation table walks and cache maintenance operations.

Chapter 6 *SMMU Performance Monitors Extension*

Read this for information about OPTIONAL SMMU support for performance monitoring functionality.

Chapter 7 *SMMU Support for Two Security States*

Read this for information about OPTIONAL SMMU support for two Security states.

Chapter 8 *SMMU Address Space*

Read this for information about the SMMU register address map in terms of the SMMU address spaces.

Chapter 9 *SMMU Global Register Space 0*

Read this for information about the contents of Global Register Space 0. This address space provides high-level control of the SMMU resources.

Chapter 10 *SMMU Global Register Space 1*

Read this for information about the contents of Global Register Space 1. In addition to providing high-level SMMU resource control, this space accommodates the number of addresses in the global register space exceeding the capacity of a single memory page, when the page size is 4KB.

Chapter 11 *SMMU IMPLEMENTATION DEFINED Address Space*

Read this for information about address space reserved for IMPLEMENTATION DEFINED purposes.

Chapter 12 *SMMU Performance Monitors Extension Register Map*

Read this for information about the recommended memory-mapped and external debug interface to the Performance Monitors Extension.

Chapter 13 *The Security State Determination Address Space*

Read this for information about the address space used by the Security State Determination part of the translation process.

Chapter 14 *Extended Stream Matching Extension*

Read this for information about the optional extension that supports up to 1024 Stream Match Register Groups.

Chapter 15 *StreamID Compressed Indexing Extension*

Read this for information about StreamID compressed indexing extension.

Chapter 16 *Stage 1 Translation Context Bank Format*

Read this for information about the stage 1 translation context bank format.

Chapter 17 *Stage 2 Translation Context Bank Format*

Read this for information about the stage 2 translation context bank format.

Appendix A *Register Names*

Read this for information about differences in register names between SMMU architecture versions.

Glossary Read this for definitions of some terms used in this specification.

Conventions

The following sections describe conventions that this book can use:

- *Typographic conventions*
- *Register names*
- *Numbers*
- *Pseudocode descriptions*.

Typographic conventions

The typographical conventions are:

italic	Introduces special terminology, and denotes citations.
bold	Denotes signal names, and is used for terms in descriptive lists, where appropriate.
`monospace`	Used for assembler syntax descriptions, pseudocode, and source code examples. Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.
SMALL CAPITALS	Used for a few terms that have specific technical meanings, and are included in the glossary.
Colored text	Indicates a link. This can be: • a URL, for example `http://infocenter.arm.com` • a cross-reference, that includes the page number of the referenced information if it is not on the current page, for example, *Pseudocode descriptions* • a link, to a chapter or appendix, or to a glossary entry, or to the section of the document that defines the colored term, for example *Translation context bank*.

Register names

In a register name, *s* denotes the presence or absence of the Secure register prefix, S. For example, in an implementation that supports two Security states, the register SMMU_*s*ACR is implemented as both:

- the Secure register SMMU_SACR
- the Non-secure register SMMU_ACR.

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 manual uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a `monospace` font.

Additional reading

This section lists relevant publications from ARM and third parties.

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

ARM publications

See the following documents for other information.

- *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition* (ARM DDI 0406).
- *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile* (ARM DDI 0487).
- *ARM® Architecture Reference Manual Supplement, ARMv8.1, for ARMv8-A architecture profile* (ARM DDI 0557).
- *CoreSight™ Architecture Specification* (ARM IHI 0029).

Feedback

ARM welcomes feedback on its documentation.

Feedback on this manual

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

- The title.
- The number, ARM IHI 0062D.c.
- The page numbers to which your comments apply.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

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

Chapter 1
Introduction

This chapter provides an introduction to the ARM *System Memory Management Unit* (SMMU) architecture. It contains the following sections:

1.1 About the ARM System MMU architecture

The ARM *System MMU* (SMMU) architecture provides a flexible implementation framework for a *Memory Management Unit* (MMU) implementation, with a number of IMPLEMENTATION DEFINED options.

The architecture can be used for a system-level MMU. It supports address translation from an *input address* to an *output address*, based on address mapping and memory attribute information held in *translation tables*.

An address translation from an input address to an output address is described as a *stage* of address translation.

The SMMU architecture also supports the concept of *translation regimes*, in which a required memory access might require two stages of address translation. For example, in a virtualized processor implementation:

- An operating system defines the translation tables for its own memory accesses, and for accesses by applications running under it. It does this believing it is mapping the *virtual addresses* (VAs) used by the processor to *physical addresses* (PAs) in the physical memory system. However, it actually defines addresses in an *intermediate physical address* (IPA) memory map.
- A hypervisor defines the translation tables that translate the IPAs for a particular *guest operating system* to PAs.

This means that any memory access by a Guest OS, or by an application, requires two stages of translation, that together define a single translation regime:

- Stage 1, from VA to IPA.
- Stage 2, from IPA to PA.

Within this system, the hypervisor must also define the required translation tables for its own memory accesses. These are in a separate translation regime, with only one stage of translation in which the stage 1 translation maps VAs to PAs.

A single stage of address translation can require multiple translation table lookups. In this case, each translation table lookup is described as a *level* of address lookup.

An implementation of the ARM SMMU architecture can provide:

- Multiple transaction contexts that apply to specific streams of transactions.
- Single or two stage translation.
- For any stage of translation, multiple levels of address lookup, to provide fine-grained memory control.
- Fault handling, logging, and signaling functionality.
- Debug and OPTIONAL performance monitoring functionality.

1.1.1 Debug support

The SMMU architecture does not require the provision of debug support features. However, if an implementation supports two Security states, a Non-secure debug agent must not be able to read any data relating to Secure transaction handling.

1.1.2 TLB visibility

ARM strongly recommends that an implementation provides a mechanism to read the content of any *Translation Lookaside Buffer* (TLB) structure in the implementation. The method by which the SMMU provides this feature is IMPLEMENTATION DEFINED, but it must ensure that Non-secure resources cannot access Secure TLB entries. The SMMU global register map reserves space to provide access to such a function. See Chapter 9 *SMMU Global Register Space 0* for more information.

1.1.3 SMMU architecture version

This specification defines version 2.0 of the SMMU architecture (SMMUv2), and also describes version 1.0 of the architecture (SMMUv1).

1.1.4 Changes in version 2.0 of the SMMU architecture

Version 2.0 of the SMMU architecture contains the following changes and additions to version 1.0:

- A requirement that any SMMUv1 Secure software that changes the partitioning of context banks between Secure and Non-secure be updated in accordance with *SMMUv2 backwards compatibility problem with SMMUv1 on page 1-20*.
- Support for the address translations required by both Execution states of ARMv8 processors, where the AArch32 state processes addresses in 32-bit registers, and the AArch64 state processes addresses in 64-bit registers. This means SMMUv2 supports the following *translation schemes*:
  - AArch32 Short-descriptor.
  - AArch32 Long-descriptor.
  - AArch64.
- Modified address translation and TLB invalidate registers, and new SMMUv2-only registers. See.
  - Chapter 9 *SMMU Global Register Space 0*.
  - Chapter 16 *Stage 1 Translation Context Bank Format*.
  - Chapter 17 *Stage 2 Translation Context Bank Format*.
- Changed status of address translation registers. These registers are OPTIONAL in SMMUv2. See *Address translation registers in SMMUv2 on page 4-126*.
- New SMMU_CBA2Rn register that extends the configuration attributes for the translation context bank. See *SMMU_CBA2Rn, Context Bank Attribute Registers on page 10-247*.
- Support for *Translation Table Base Register* (TTBR) determination for Secure and Non-secure translation context banks. See *The translation context bank table on page 2-75*.
- Support for a 16KB and 64KB *translation granule* size. See *SMMU translation schemes on page 1-28* and *SMMU_IDR2 on page 9-186*.
- Support for Monitor context for the Secure EL3 translation regime. See *Monitor contexts (MONC) on page 2-89*.
- Support for E2H context. See *E2H contexts (E2HC) on page 2-90*.
- Modified context interrupt behavior. See *Context interrupts on page 3-100*.
- Modified invalid context fault behavior. See *Global faults on page 3-114*.
- Modified configuration access fault behavior. See *Configuration access on page 3-119* and *SMMU_SCR1, Secure Configuration Register 1 on page 9-226*.
- Modified external fault behavior. See *External faults on page 3-120*.
- Modified MemAttr encoding. See *Memory attribute, MemAttr on page 9-179* and *SMMU_CBn_MAIRm, Memory Attribute Indirection Registers on page 16-304*.
- Renaming of SMMU_CBn_ TTBCR registers to SMMU_CBn_TCR registers.
- New SMMU_CBn_TCR2 register that extends the SMMU_CBn_TCR functionality. See *SMMU_CBn_TCR2, Translation Control Register 2 on page 16-336*.
- Modified SMMU_CBn_RESUME behavior. See *SMMU_CBn_RESUME, Transaction Resume register on page 16-314*.
- New SMMU_CBn_IPAFAR register that records the IPA for transactions that fault during stage 2 translation. See *SMMU_CBn_IPAFAR, IPA Fault Address Register on page 17-350*.
- Modified behavior when certain register fields are encoded as `Reserved`. See *Reserved memory type and shareability attributes on page 2-62*.

1.1.5 SMMUv2 backwards compatibility problem with SMMUv1

SMMUv1 compliant Non-secure software running with a non-existent or SMMUv2 compliant Secure software requires no change.

SMMUv1 compliant Secure software that changes the partitioning of the context banks between Secure and Non-secure must be updated for SMMUv2. The change required should be confined to the code that changes the partitioning, that is, the write of SMMU_SCR1.NSNUMCBO.

See *Resource allocation on page 7-158* for more information.

1.2 About the SMMU

The interfaces to an ARM SMMU are IMPLEMENTATION DEFINED, but the description of the SMMU in this manual is based on the implementation model summarized in this section,

Figure 1-1 shows the implementation of an ARM SMMU in the memory system.

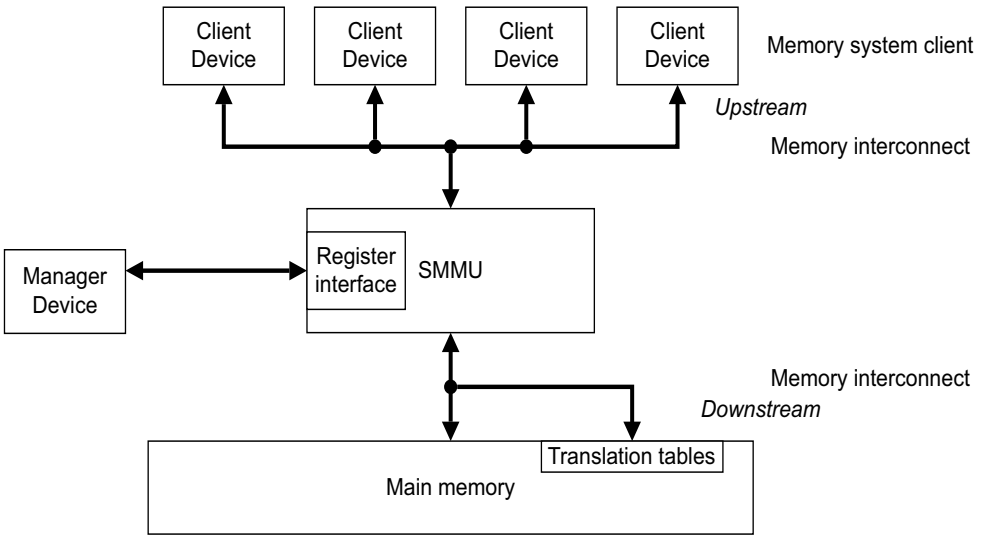


Figure 1-1 An ARM SMMMU in the memory system

In this implementation model:

- One or more *Client devices* connect through the memory interconnect to the SMMU.
- Client devices are described as being *upstream* of the SMMU. The connection between the SMMU and the client devices is the *upstream bus*.
- The SMMU connects to the rest of the memory system, through to main memory.
- The rest of the memory system is described as being *downstream* of the SMMU. The connection between the SMMU and the rest of the memory system is the *downstream bus*.
- A client device issues a transaction request to the SMMU. The SMMU processes that transaction, and returns a response to the client.

  A simple description of a possible usage model for the SMMU is that client device makes a memory access that is routed to the SMMU. The SMMU performs the address translation required for the access, using translation tables in memory, and makes the required access permission and attribute checks:

  — If the translation is successful and the access is valid, the SMMU performs the access and returns the result to the client.

  — Otherwise, the SMMU returns fault information to the client.

  The client device might not be aware that the SMMU is handling its access to the memory system.

In addition to these memory system connections to the SMMU, the SMMU provides a memory-mapped register interface. A *manager device* uses this interface to configure and control the SMMU.

Note

This specification assumes that the *manager device* is independent of any *client device*, although this is not a requirement of the SMMU architecture.

This specification defines:

- The processing of transaction requests by the SMMU.

- The memory-mapped register interface to the SMMU.

1.3 ARM PE Exception levels and Execution states

The ARMv8 exception model defines the Exception level ELn, where n can have the values 0-3. Software execution privilege increases with increase in the value of n, with EL0 software having the lowest level of privilege. Execution at EL0 is described as unprivileged execution. It is IMPLEMENTATION DEFINED whether a processor implementation includes EL2 or EL3. The typical use of the different Exception levels is:

EL0 Application software. Secure or Non-secure state.

EL1 Operating system. Secure or Non-secure state.

EL2 Hypervisor. Non-secure state only.

EL3 Secure monitor.

The permitted processor Exception levels depend on the processor Security state, as Table 1-1 shows.

Table 1-1 Exception level implementation by Security state

Non-secure	Secure
-	EL3
EL2	-
EL1	EL1
EL0	EL0

ARMv8 defines AArch64 and AArch32 Execution states that define the supported instruction set, and whether the processor uses 64-bit or 32-bit PC, LR, SP, and general-purpose registers.

Based on the AArch32 and AArch64 Execution states and the associated translation table formats, the SMMU architecture defines the following *translation schemes*:

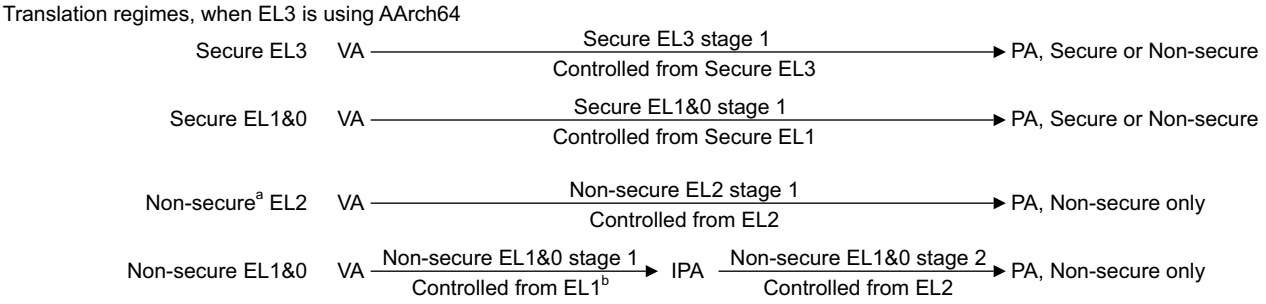
- AArch32 Short-descriptor.
- AArch32 Long-descriptor.
- AArch64.

The remainder of this chapter provides more information about the SMMU translation schemes.

1.4 ARM translation regimes

The ARM architecture supports different translation regimes and stages for AArch32 and AArch64. Figure 1-2 shows the ARM architecture translation regimes for memory accesses made from AArch64 state, where.

- VA is Virtual Address.
- PA is Physical Address.
- IPA is Intermediate Physical Address.



a. The SMMU also includes a Secure EL2 translation regime

b. Or higher

Figure 1-2 ARM processor architecture AArch64 translation regimes and stages

In Figure 1-2:

- The Non-secure EL1&0 translation regime comprises two stages of translation.
- The other translation regimes comprise only a single stage of translation.

Translation regimes for accesses from AArch32 state describes how the translation regimes differ for memory accesses from AArch32 state, and the different terminology used to describe these regimes.

1.4.1 Translation regimes for accesses from AArch32 state

In the ARM processor architecture, the behavior of a PE that is executing in AArch32 state is defined in relation to different PE *modes*. Most of these modes have the level of execution privilege that is appropriate to an operating system. In a processor implementation that includes EL3, if EL3 can use AArch32 state or AArch64 state, the Exception level of these modes can depend on whether EL3 is using AArch32 or AArch64, as Table 1-2 shows.

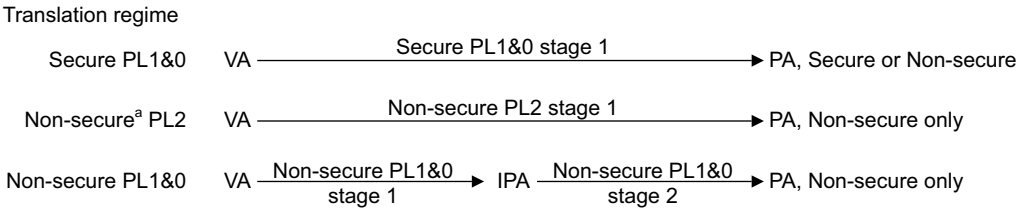
Table 1-2 Mapping of AArch32 PE modes to Exception levels

PE mode	Non-secure state	Secure state		Privilege level
		EL3 using AArch64	EL3 using AArch32	
User	EL0	EL0	EL0	PL0
System, Supervisor, Abort, Undefined, IRQ, FIQ	EL1	EL1	EL3	PL1
Hyp[a]	EL2	-	-	PL2
Monitor[b]	-	-	EL3	PL1

a. Implemented only in Non-secure state, and only present when EL2 is using AArch32.

b. Implemented only in Secure state, and only present when EL3 is using AArch32.

To provide descriptions of the AArch32 translation regimes that are independent of whether EL3 is using AArch32 or AArch64, these translation regimes are described in terms of *privilege levels*, PL0-PL2, as Figure 1-3 shows.



a. The SMMU also includes a Secure EL2 translation regime

Figure 1-3 ARM processor architecture AArch32 translation regimes and stages

Note

The regime that is affected by how the mapping of PE modes to Exception level is determined by the current Execution state is the Secure PL1&0 translation regime. If this was described in terms of Exception levels, it would be:

- The Secure EL1&0 regime when EL3 is using AArch64.
- The Secure EL3&0 regime when EL3 is using AArch32.

The SMMU controls for the Secure PL1&0 translation regime are those defined for the Secure EL1&0 translation regime.

For the Non-secure PL1&0 translation regime, the second stage of translation, that maps the IPA from stage 1 to a PA, might be:

- An AArch32 translation stage, as shown in Figure 1-3.
- An AArch64 translation stage, as shown in Figure 1-2 on page 1-24.

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

1.4.2 SMMU register controls of the translation regimes

The different translation regimes have associated SMMU registers, that are accessible at the corresponding Exception level, as Table 1-3 on page 1-26 shows.

Table 1-3 Translation regime control of SMMU registers

Translation regime	Register	Description	Notes
Secure EL3 Secure EL1&0	SMMU_CBA2Rn	*SMMU_CBA2Rn, Context Bank Attribute Registers on page 10-247*	Register applies to Secure context banks
	SMMU_CBARn	*SMMU_CBARn, Context Bank Attribute Registers on page 10-241*	Register applies to Secure context banks
	SMMU_CBFRSYNRAn	*SMMU_CBFRSYNRAn, Context Bank Fault Restricted Syndrome Register A on page 10-249*	-
	SMMU_SCR0	*SMMU_SCR0, Secure Configuration Register 0 bit assignments on page 9-190*	-
	SMMU_SCR1	*SMMU_SCR1, Secure Configuration Register 1 on page 9-226*	-
	SMMU_SGATS*	*SMMU Global Register Space 0 register descriptions on page 9-181*	Secure global address translation registers
	SMMU_SGF*	*SMMU Global Register Space 0 register descriptions on page 9-181*	Secure global fault registers
	SMMU_SGPAR	*SMMU_sGPAR, Global Physical Address Register on page 9-214*	-
	SMMU_NSGATS*	*Secure alias for Non-secure registers on page 9-177*	Secure aliases for Non-secure copies of Global address translation registers
	SMMU_NSGF*	*Secure alias for Non-secure registers on page 9-177*	Secure aliases for Non-secure copies of Global fault registers
	SMMU_NSGPAR	*Secure alias for Non-secure registers on page 9-177*	Secure alias for Non-secure copy of SMMU_sGPAR
	SMMU_sTLBGSTATUS	*SMMU_sTLBGSTATUS, Global TLB Status register on page 9-218*	Secure alias for Non-secure copies of Global TLB registers
	SMMU_sTLBGSYNC	*SMMU_sTLBGSYNC, Global Synchronize TLB Invalidate on page 9-219*	
	SMMU_SSDRn	*SMMU SSD address space on page 13-272*	Security state determination registers
	SMMU_STLBI	Chapter 9 *SMMU Global Register Space 0*	Secure TLB Invalidate registers
	SMMU_S2CRn	*SMMU_S2CRn, Stream-to-Context Register on page 9-220*	Registers apply to Secure Stream Match Register groups
	SMMU_SMRn	*SMMU_SMRn, Stream Match Register on page 9-231*	Registers apply to Secure Stream Match Register groups
Secure EL3	SMMU_CBn_*	Chapter 17 *Stage 2 Translation Context Bank Format*	Monitor context, Secure only
Secure EL2	SMMU_CBn_*	Chapter 17 *Stage 2 Translation Context Bank Format*	Hypervisor context, Secure and Non-secure

Table 1-3 Translation regime control of SMMU registers (continued)

Translation regime	Register	Description	Notes
Non-secure EL2	SMMU_CR0	*SMMU_CR0, Non-secure Configuration Register 0 bit assignments on page 9-195*	-
	SMMU_GATS*	*SMMU Global Register Space 0 register descriptions on page 9-181*	Non-secure global address translation registers
	SMMU_GF*		Non-secure global fault registers
	SMMU_GPAR	*SMMU_sGPAR, Global Physical Address Register on page 9-214*	-
	SMMU_CBARn	*SMMU_CBARn, Context Bank Attribute Registers on page 10-241*	Register applies to Non-secure context banks
	SMMU_CBA2Rn	*SMMU_CBA2Rn, Context Bank Attribute Registers on page 10-247*	Register applies to Non-secure context banks
	SMMU_S2CRn	*SMMU_S2CRn, Stream-to-Context Register on page 9-220*	Registers apply to Non-secure Stream Match Register groups
	SMMU_SMRn	*SMMU_SMRn, Stream Match Register on page 9-231*	Registers apply to Non-secure Stream Match Register groups
	SMMU_CBn_*	Chapter 17 *Stage 2 Translation Context Bank Format*	Registers apply to stage 2 translation context banks
Secure EL1&0	SMMU_CBn_*	Chapter 16 *Stage 1 Translation Context Bank Format*	For Secure context banks
Non-secure EL1&0	SMMU_CBn_*		For Non-secure context banks

1.5 SMMU translation schemes

The SMMUv2 translation schemes support address translations required by both the AArch32 and AArch64 Execution states, for each translation regime, as follows:

AArch32 Short-descriptor translation scheme

This translation scheme:

- Uses 32-bit entries, or *descriptors*, in its translation tables.
- Is compatible with the ARMv7 architecture.
- Supports the translation of VAs of up to 32 bits.
- Supports output addresses of up to 32 bits, or up to 40 bits with loss of granularity.
- Can be used only from the AArch32 processor Execution state.
- Can be used only for stage 1 translation.

AArch32 Long-descriptor translation scheme

This translation scheme:

- Uses 64-bit descriptors in its translation tables.
- Is added to the ARMv7 architecture by the *Large Physical Address Extension* (LPAE).
- Supports the translation of VAs of up to 32 bits.
- Supports output addresses of up to 40 bits, that can be IPAs or VAs.

AArch64 translation scheme

This translation scheme:

- Uses 64-bit descriptors in its translation tables.
- Is defined by the ARMv8 architecture.
- Supports the translation of VAs of up to 49 bits.
- Supports output addresses of up to 48 bits, as two independent address ranges.

In the ARM architecture, a *translation granule* is the smallest region of input address space that an SMMU implementation can be configured to support. A translation granule defines both:

- The maximum size of a single translation table.
- The memory page size, that is, the granularity at which attributes can be assigned to memory regions.

The AArch32 translation schemes use a translation granule of 4KB, meaning that the translation tables can define the attributes of memory regions at a granularity of 4KB.

Although the AArch32 Short-descriptor translation scheme has a translation granule of 4KB, it can support output addresses of more than 32 bits by using page sizes larger than 4KB.

The AArch32 Long-descriptor translation scheme:

- Has a translation granule of 4KB for stage 1 translations.
- By default, has a translation granule of 4KB for stage 2 translations.

The AArch64 translation scheme supports a translation granule of 4KB, 16KB, or 64KB.

For EL1&0 stage 1 translations, the ARM architecture supports two VA subranges, which are determined by the SMMU_CBn_TCR.T0SZ and SMMU_CBn_TCR.T1SZ size fields:

- The AArch32 translation schemes provide these two subranges by splitting the available 32-bit VA range.
- In the AArch64 translation scheme, the two subranges are independent ranges of up to 48-bits, where:
  - The lower VA range runs from `0x0000_0000_0000_0000` to (`Size indicated by T0SZ`).
  - The upper VA range runs from (`0xFFFF_FFFF_FFFF_FFFF - (Size indicated by T1SZ)`) to `0xFFFF_FFFF_FFFF_FFFF`.

  There is always a gap between the two VA subranges.

  The VA model for the AArch64 translation scheme can be described as using a 49-bit VA range, where VA[48] indicates the VA subrange for the address.

For more information, see *Defining the VA subranges for stage 1 translations on page 1-30*.

Table 1-4 shows the AArch64 translation regimes. All of these regimes use the AArch64 translation scheme, for all translations.

Table 1-4 AArch64 translation regimes

Translation regime	Translation context	Note
Secure EL3	Stage 1 context with stage 2 bypass	Monitor context only. See *Monitor contexts (MONC) on page 2-89* for more information.
Secure EL1		-
Non-secure EL2		*Hypervisor context* (HYPC) only. See *Hypervisor contexts (HYPC) on page 2-86* for more information.
Secure EL2[a]		
Non-secure EL1&0, no EL2[b]		-
Non-secure EL1&0, with EL2[c]	Stage 1 followed by stage 2	For stage 2, VMSAv8 Long-descriptor
Non-secure IPA address space[d]	Stage 2	Applies to device-based stage 1 translation and IPA generation

a. This translation regime does not correspond to an ARM processor architecture translation regime.
b. The implementation does not include EL2.
c. The implementation includes EL2.
d. This translation regime does not correspond to an ARM processor architecture translation regime, but is equivalent to the stage 2 translation of the Non-secure EL1&0 translation regime.

Table 1-5 shows the AArch32 translation regimes, and translation schemes that can be used with each regime

Table 1-5 AArch32 translation regimes

Translation regime	Translation context	Note
Secure EL1	Stage 1 context with stage 2 bypass	Either of: • AArch32 Short-descriptor. • AArch32 Long-descriptor.
Non-secure EL2		AArch32 Long-descriptor, hypervisor context (HYPC) only. See *Hypervisor contexts (HYPC) on page 2-86* for more information.
Secure EL2[a]		
Non-secure EL1&0, no EL2[b]		Either of: • AArch32 Short-descriptor. • AArch32 Long-descriptor.
Non-secure EL1&0, with EL2[c]	Stage 1 followed by stage 2	Stage 1 can use either AArch32 translation scheme. Stage 2 can use either of: • AArch32 Long-descriptor translation scheme. • AArch64 translation scheme.
Non-secure IPA address space[d]	Stage 2	Applies to device-based stage 1 translation and IPA generation

a. This translation regime does not correspond to an ARM processor architecture translation regime.
b. The implementation does not include EL2.

c. The implementation includes EL2.

d. This translation regime does not correspond to an ARM processor architecture translation regime, but is equivalent to the stage 2 translation of the Non-secure EL1&0 translation regime.

1.5.1 Defining the VA subranges for stage 1 translations

How the VA subranges are defined depends on the address translation scheme, as described in the following subsections:

- *VA subrange definition, AArch32 Short-descriptor translation scheme*.
- *VA subrange definition, AArch32 Long-descriptor translation scheme on page 1-31*.
- *VA subrange definition, AArch64 translation scheme on page 1-33*.

VA subrange definition, AArch32 Short-descriptor translation scheme

In the PL1&0 translation regime, the VA range is split into two subranges as shown in Figure 1-4 on page 1-31. When using the Short-descriptor translation table format, the value of SMMU_CBn_TCR.T0SZ indicates the number of most significant bits of the input VA that determine whether TTBR0 or TTBR1 holds the required translation table base address, as follows:

- If T0SZ == 0, then use TTBR0. Setting SMMU_CBn_TCR.T0SZ to zero disables use of a second set of translation tables.
- If T0SZ > 0, then:
  - If bits[31:32-T0SZ] of the input VA are all zeros, then use SMMU_CBn_TTBR0.
  - Otherwise, use SMMU_CBn_TTBR1.

Table 1-6 shows how the value of T0SZ determines the lowest address translated using SMMU_CB*n*_TTBR1, and the size of the level 1 translation table addressed by SMMU_CB*n*_TTBR0.

Table 1-6 Effect of SMMU_CB*n* _TCR.T0SZ on address translation, Short-descriptor format

SMMU_CB*n*_TCR.T0SZ	First address translated with SMMU_CB*n*_TTBR1	TTBR0 table	
		Size	Index range
0b000	TTBR1 not used	16KB	VA[31:20]
0b001	0x80000000	8KB	VA[30:20]
0b010	0x40000000	4KB	VA[29:20]
0b011	0x20000000	2KB	VA[28:20]
0b100	0x10000000	1KB	VA[27:20]
0b101	0x08000000	512 bytes	VA[26:20]
0b110	0x04000000	256 bytes	VA[25:20]
0b111	0x02000000	128 bytes	VA[24:20]

When SMMU_CBn_TCR.T0SZ is nonzero, the size of the translation table addressed by SMMU_CB*n*_TTBR1 is 16KB.

Figure 1-4 shows how the value of SMMU_CBn_TCR.T0SZ controls the boundary between VAs that are translated using SMMU_CB*n*_TTBR0, and VAs that are translated using SMMU_CB*n*_TTBR1.

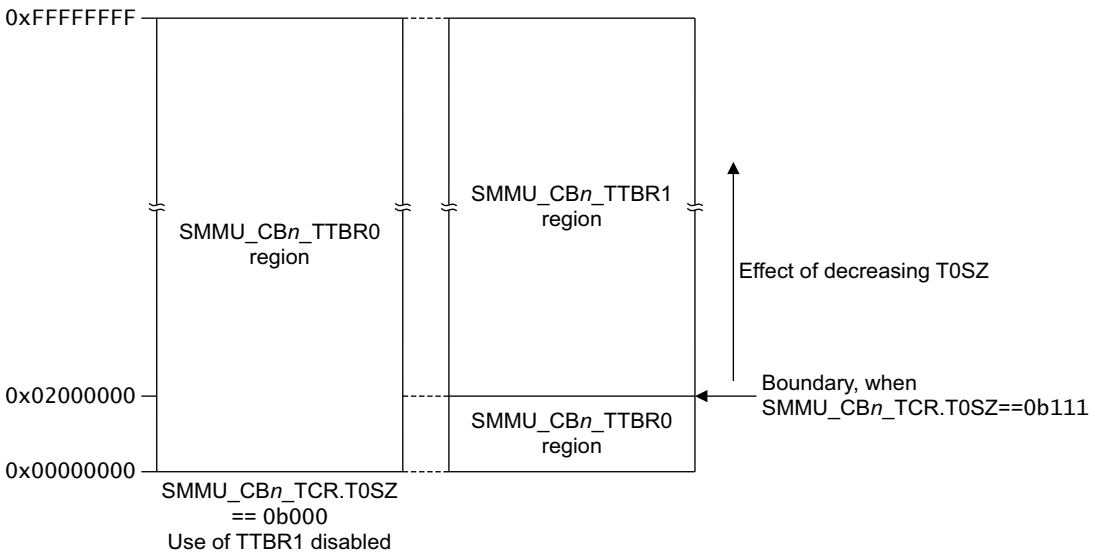


Figure 1-4 How SMMU_CB*n*_TCR.T0SZ controls the boundary between the TTBRs, Short-descriptor format

VA subrange definition, AArch32 Long-descriptor translation scheme

For the PL1&0 translation regime, the VA range is split into two subranges as shown in Figure 1-5 on page 1-32 or Figure 1-6 on page 1-32. When an AArch32 PL1&0 stage 1 address translation that uses the Long-descriptor translation scheme is enabled, SMMU_CB*n*_TTBR0 is always used. If SMMU_CB*n*_TTBR1 is also used, then:

- SMMU_CB*n*_TTBR1 is used for the top part of the input address range.
- SMMU_CB*n*_TTBR0 is used for the bottom part of the input address range.

The SMMU_CBn_TCR.T0SZ and SMMU_CBn_TCR.T1SZ size fields control the use of SMMU_CBn_TTBR0 and SMMU_CBn_TTBR1, as

Table 1-7 Use of SMMU_CBn_TTBR0 and SMMU_CBn_TTBR1, AArch32 Long-descriptor format

		Input address range using:	
T0SZ	**T1SZ**	**SMMU_CBn_TTBR0**	**SMMU_CBn_TTBR1**
0b000	0b000	All addresses	Not used
M[a]	0b000	Zero to $(2^{(32-M)}-1)$	2^{32-M} to maximum input address
0b000	*N*[a]	Zero to $(2^{32}-2^{(32-N)}-1)$	$2^{32}-2^{(32-N)}$ to maximum input address
M[a]	*N*[a]	Zero to $(2^{(32-M)}-1)$	$2^{32}-2^{(32-N)}$ to maximum input address

a. *M*, *N* must be greater than 0. The maximum possible value for each of T0SZ and T1SZ is 7.

For stage 1 translations, the input address is always a VA, and the maximum possible VA is $(2^{32}-1)$.

When address translation is using the AArch32 Long-descriptor translation scheme:

- Figure 1-5 shows how, when SMMU_CBn_TCR.T1SZ is zero, the value of SMMU_CBn_TCR.T0SZ controls the boundary between VAs that are translated using SMMU_CB*n*_TTBR0, and VAs that are translated using SMMU_CB*n*_TTBR1.

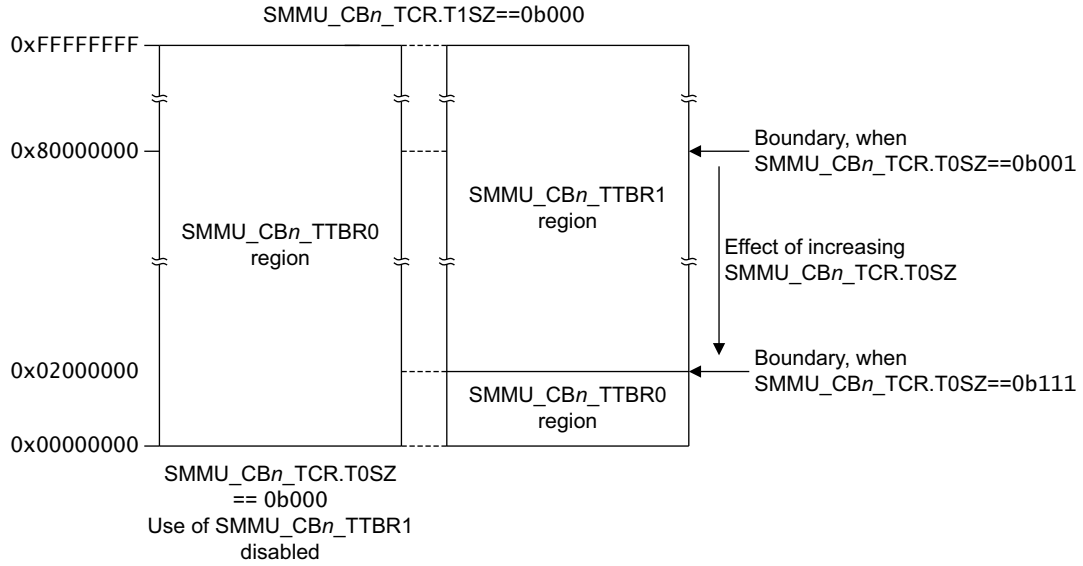


Figure 1-5 Control of TTBR0 and TTBR1 regions, when SMMU_CB*n*_TCR.T1SZ is zero

- Figure 1-6 shows how, when SMMU_CBn_TCR.T1SZ is nonzero, the values of SMMU_CBn_TCR.T0SZ and SMMU_CBn_TCR.T1SZ control the boundaries between VAs that are translated using SMMU_CB*n*_TTBR0, and VAs that are translated using SMMU_CB*n*_TTBR1.

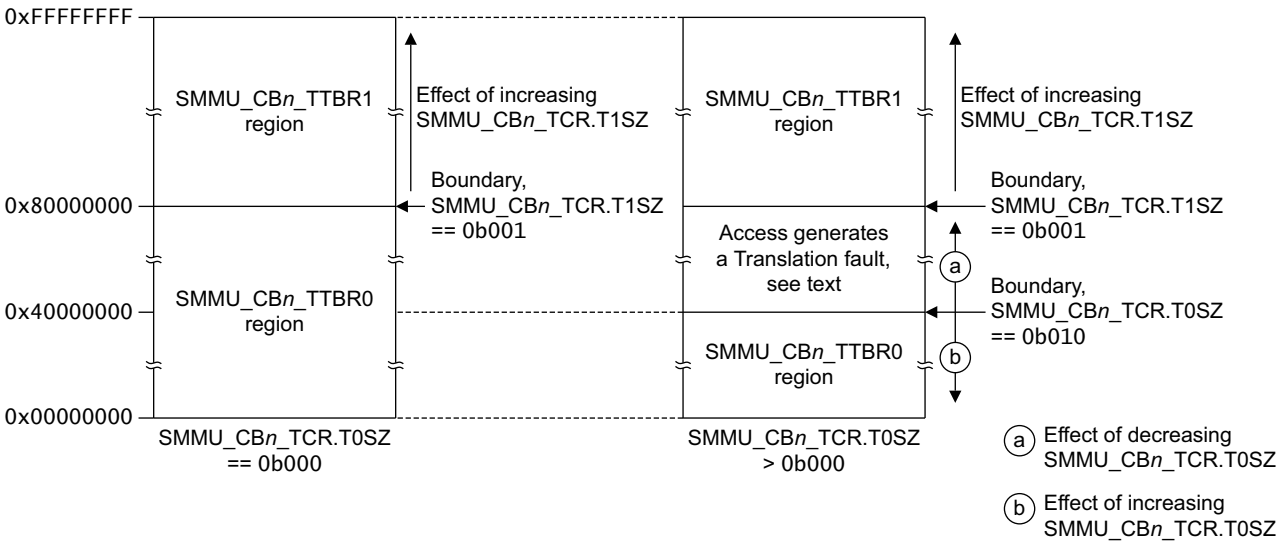


Figure 1-6 Control of TTBR0 and TTBR1 regions when SMMU_CB*n*_TCR.T1SZ is nonzero

When SMMU_CBn_TCR.{T0SZ, T1SZ} are both nonzero:

— If both fields are set to `0b001`, the boundary between the two regions is 0x80000000. This is identical to having T0SZ set to `0b000` and T1SZ to `0b001`.

— Otherwise, the SMMU_CB*n*_TTBR0 and SMMU_CB*n*_TTBR1 regions are contiguous. In this case, any attempt to access an address that is in that gap between the SMMU_CB*n*_TTBR0 and SMMU_CB*n*_TTBR1 regions generates a Translation fault.

VA subrange definition, AArch64 translation scheme

For the EL1&0 translation regime, the VA range is split into two subranges as shown in Figure 1-7, and:

- SMMU_CB*n*_TTBR0 points to the initial translation table for the lower VA subrange, that starts at address `0x0000_0000_0000_0000`. SMMU_CBn_TCR.T0SZ determines the region size, which is $2^{(64-T0SZ)}$ bytes.
- SMMU_CB*n*_TTBR1 points to the initial translation table for the upper VA subrange, that runs up to address `0xFFFF_FFFF_FFFF_FFFF`. SMMU_CBn_TCR.T1SZ determines the region size, which is $2^{(64-T1SZ)}$ bytes.

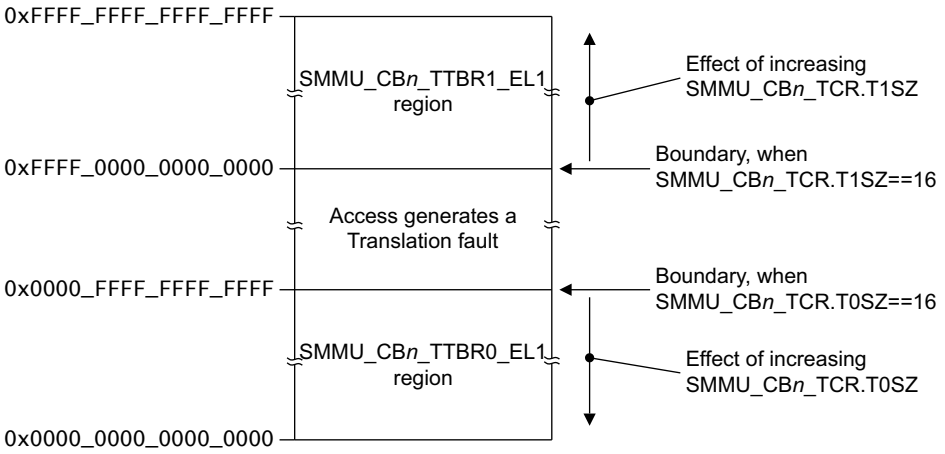


Figure 1-7 AArch64 TTBR boundaries and VA ranges

Which SMMU_CBn_TTBRm is used depends only on the VA presented for translation:

- If the top bits of the VA are zero, then SMMU_CBn_TTBR0 is used.
- If the top bits of the VA are one, then SMMU_CBn_TTBR1 is used.

Note

If the value of SMMU_CBn_TCR2.TBI*m* is 1, then for an address in SMMU_CBn_TTBRm region, bits[63:56] of the 64-bit value are excluded from the requirement that the top bits of the VA are either all 0 or all 1.

1.5.2 Differences between the ARM architecture and SMMU translation schemes

Although the SMMU architecture is based on the ARM translation regimes, differences exist between the SMMU architecture and the ARM architecture. This section describes these differences.

Transient allocation hint

The ARM architecture does not support the transient allocation hint when using a Short-descriptor translation table format. An ARM processor using this format marks all transactions as non-transient.

In SMMUv2, when using the AArch32 Short-descriptor translation scheme, the attributes provided by the incoming transaction, which might be modified by SMMU_S2CRn, provide the transient allocation hint for stage 1 attributes.

HYPC and MONC

The SMMUv2 definition of *HYPC* (hypervisor context) and *MONC* (Monitor context) banks closely match the ARM Non-secure EL2 and Secure EL3 address translation regimes. However, this section describes some differences.

The ARM architecture TTBR does not permit translation table walks to be disabled, and provides no equivalent to the SMMU_CBn_TCR.EPD, the Translation Walk Disable bits. However, the SMMU architecture does not ignore SMMU_CBn_TCR.EPD0, and ARM recommends that this is set to 0.

In SMMUv2, subject to the restrictions described in *Hypervisor contexts (HYPC) on page 2-86*, software can configure HYPC banks as Secure. This means that SMMU HYPC banks must adhere to the appropriate SMMU_CBn_TCR.NSCFG0 or SMMU_CBn_TCR2.NSCFG0 settings.

It is IMPLEMENTATION DEFINED whether Secure Hypervisor TLB entries are subject to broadcast TLB maintenance operations. It is also IMPLEMENTATION DEFINED whether SMMU_SCR0.PTM or SMMU_CR0.PTM controls whether they are affected by broadcast TLB maintenance. For this reason, ARM strongly recommends that software using Secure Hypervisor entries always maintains these TLB entries directly using the register interface.

See *Hypervisor contexts (HYPC) on page 2-86* and *Monitor contexts (MONC) on page 2-89* for more information.

Translation stages

In both SMMUv1 and SMMUv2, conceptually a translation regime that starts with a stage 1 translation is always followed by a second stage of translation, even for the following ARM processor architecture translation regimes that have only a single stage:

- Hypervisor
- Monitor
- Secure EL1&0 or PL1&0.
- Non-secure EL1&0, when there is no hypervisor.

The SMMU equivalent to these single stage translation regimes is where SMMU_CBARn specifies a Stage 1 context with stage 2 bypass format for a context bank. This format provides an attribute-only transformation stage, where SMMU_CBARn provides the stage 2 attributes. This attribute-only stage applies also to any memory accesses made as part of a translation table walks performed for a stage 1 translation.

Note

- For Non-secure, non-HYPC banks a *virtual machine identifier* (VMID) is still required, even if the context bank is stage 1 with stage 2 bypass, as in the ARM architecture.
- The SMMU Non-secure IPA address space translation scheme has only a single stage of translation, that is a stage 2 translation.

If a Guest OS uses multiple VMIDs, the hypervisor must virtualize the VMID values. To minimize the virtualization effort on the hypervisor, ARM recommends that a Guest OS uses a single fixed VMID value of zero.

Shared translation tables

When the SMMU and the processor share translation tables, and the signaling of an incoming SMMU transaction request identifies it as an instruction access, software must treat regions that are marked as *Execute-never* (XN) or *Execute-only* (XO) in one of the following ways:

- XN and XO regions apply to both execution by the processor and by peripheral devices.
- XN and XO regions apply only to processor execution.

When XN and XO regions apply only to the processor, software can use the SMMU_S2CRn.INSTCFG field to configure the SMMU to treat such transactions as data. This field can be used to override the attributes signaled as part of the transaction request.

Note

- SMMUv1 does not support XO marking.
- The SMMU performs the SMMU_SCR1.SIF protection check after applying the INSTCFG attribute. The SMMU_SCR1.SIF protection check therefore has no effect when INSTCFG overrides the marking of a region marked as instruction.
- Device literal pools are classed as data. Where a peripheral issues both instruction and data transactions, software must ensure that setting a region as XO does not prevent the peripheral from accessing device literal pools.

When the SMMU shares translation tables with the processor, it is possible for the SMMU TLB to contain obsolete descriptors. See the stage 1 SMMU_CBn_SCTLR.AFFD, or the stage 2 SMMU_CBn_SCTLR.AFFD bit description for more information.

Cache maintenance operations

In both the SMMU and the ARM architecture, it is possible to disable EL0 cache maintenance operations. However, in the SMMU, this control applies regardless of whether SMMU operation is enabled. See the UCI bit description in *SMMU_CBn_SCTLR, System Control Register on page 16-315*.

TLB maintenance operations

SMMU_STLBIALL affects all Secure entries, except that it is IMPLEMENTATION DEFINED whether it affects Secure Hypervisor entries, whereas the equivalent operation on the PE will not affect EL3 entries.

Address Size Fault in AArch32 mode

In SMMUv2, an Address Size Fault is always taken if:

- The IPA is not representable in the available hardware to deal with intermediate physical addresses as specified by SMMU_IDR2.IAS.
- The PA is not representable on the downstream bus of the SMMU as specified by SMMU_IDR2.OAS. See *Conditions where a stage 1 output address generates an address size fault on page 3-105*.

Note

This differs from the conditions for VMSAv8-32 Address Size faults in the ARM processor architecture.

1.6 SMMU address support

The SMMUv2 architecture supports:

- VAs of up to.
  - 32 bits when held in 32-bit registers.
  - 49 bits when held in 64-bit registers. Of these bits, when bit[49] is valid it determines the *Translation Table Base Register* (TTBR) that is used to translate the address.
- Translation granule sizes of 4KB, 16KB, and 64KB. It is IMPLEMENTATION DEFINED which of these granule sizes are supported. The granule size defines the finest granularity of translations, but all translation schemes support translation of larger blocks of memory, of certain multiples of the granule size.
- IPAs of up to 48 bits.
- PAs of up to 48 bits.

The SMMUv1 architecture supports:

- VAs of up to 32 bits.
- IPAs of up to 40 bits.
- When using the AArch32 Short-descriptor translation scheme, PAs of up to:
  - 32 bits, with a translation granularity of 4KB, 64KB, or 1MB.
  - Optionally, 40 bits, but only with a translation granularity of 16MB.
- When using the AArch32 Long-descriptor translation scheme, PAs of up to 40 bits.

In any SMMU implementation:

- A T0SZ field, and in some translation regimes a T1SZ field, indicate the supported VA range or ranges.
- The supported IPA and PA sizes are IMPLEMENTATION DEFINED.

For more information about SMMU address support, see:

- *SMMU address size parameters*.
- *SMMU treatment of addresses on page 1-37*.
- *SMMU translation bypass conditions on page 1-37*.
- *Sign-extension of input addresses on page 1-38*.
- *Address Space Identifiers, ASIDs on page 1-38*.
- *Virtual machine identifiers, VMIDs on page 1-39*.

1.6.1 SMMU address size parameters

The SMMU_IDR2 register provides the following information about the physical hardware that is available for representing addresses within an SMMU implementation:

- The UBS field defines the physical width of the largest upstream address port of the SMMU, and represents the maximum input address to the first stage of translation.
- The IAS field defines the supported width of the SMMU hardware. The SMMU cannot represent an IPA larger than this value.
- The OAS field defines the width of the downstream address port of the SMMU. This corresponds to the maximum PA size that the SMMU supports.

The UBS, IAS, and OAS values represent the absolute maximum that the hardware can support. However, software can use various configuration mechanisms to restrict these ranges. These mechanisms are described in *SMMU treatment of addresses on page 1-37*.

If stage 1 translation is not supported, that is when SMMU_IDR0.S1TS==0, then SMMU_IDR2.UBS and SMMU_IDR2.IAS must specify the same value.

If stage 2 translation is not supported, that is when SMMU_IDR0.S2TS==0, then SMMU_IDR2.IAS and SMMU_IDR2.OAS must specify the same value.

To support fault checks, the SMMU architecture defines the following terms:

IPA size

The *IPA size* is the maximum IPA size, as defined by SMMU_IDR2.IAS.

Effective IPA size

The *effective IPA size* depends on the translation context, and is defined as:

- SMMU_IDR2.IAS, for stage 1 followed by stage 2 translations
- For stage 1 context with stage 2 bypass translations, the minimum of:
  - SMMU_IDR2.IAS.
  - SMMU_IDR2.OAS.

Output address size

The *output address size* is the maximum PA size, as defined by SMMU_IDR2.OAS.

PA size The SMMU might support more address space than a device provides. In AArch64, *PA size*, as defined by the PASize field, enables software to reduce the amount of address space to meet such requirements. The *PA size* can also restrict the operation of a process to a particular section of the available PA address space. In SMMUv2:

- SMMU_CBn_TCR2.PASize defines this size for stage 1 translations.
- SMMU_CBn_TCR.PASize defines this size for stage 2 translations.

1.6.2 SMMU treatment of addresses

Conceptually, the SMMU address handling sequence is as follows:

1. If a transaction bypasses SMMU translation before reaching a context bank, the SMMU truncates any input address that exceeds the value of SMMU_IDR2.OAS. See *SMMU translation bypass conditions*.

   In all other cases, the SMMU passes the transaction to a translation context bank, and proceeds with steps 2 and 3.

2. In certain cases, the input address is sign-extended, as described in *Sign-extension of input addresses on page 1-38*.

3. For each stage of translation, the SMMU performs checks on the input address and the output address. These checks might result in a fault being generated, Various factors affect how the SMMU performs these checks, including:

   - Whether a stage 1 or stage 2 translation context bank is used.
   - The value of SMMU_CBn_SCTLR.M for the context bank.
   - Whether a 32-bit or 64-bit descriptor format is used, as defined by the SMMU_CBA2Rn.VA64 bit.

   Translation faults on page 3-104 and *Address size faults on page 3-105* provide more information about input and output address size checks.

1.6.3 SMMU translation bypass conditions

A transaction bypasses SMMU translation when at least one of the following applies:

- The SMMU_sCR0.CLIENTPD bit is set to 1.
- The SMMU_S2CRn.TYPE is configured as Bypass mode.
- The implementation is using Stream matching, the value of SMMU_sCR0.USFCFG is 0, and either:
  - No transaction match is found in the Stream mapping table.
  - Multiple transaction matches are found in the Stream mapping table.

1.6.4 Sign-extension of input addresses

The SMMU can support an upstream bus size of either 64 bits or up to 49 bits, consisting of 48 address bits and one sign bit.

If the SMMU upstream bus is *N* bits in width and a device provides a larger VA of *M* bits in width, VA[M-1:N-1] are compressed to form VA[N-1] of the SMMU input address. The compression mechanism is IMPLEMENTATION DEFINED but the SMMU architecture requires that low-order bits[N-2:0] are preserved.

Note

Examples of compression mechanisms include:

- Ignore VA[M-1:N].
- VA[N-1] of the SMMU input address is a bitwise AND of all the bits of VA[M-1:N-1].

If bits[M-1:N-1] are not all 0s or all 1s and therefore cannot be compressed, an implementation can record and report a fault in an IMPLEMENTATION DEFINED manner. For example, an implementation where bits[M-1:N-1] are received by the SMMU might generate a UUT fault if bits[M-1:N-1] are not all 0s or all 1s.

The SMMU input address is sign-extended when all the following apply:

- A stage 1 translation context bank is used.
- The AArch64 translation scheme is selected.
- SMMU_CBn_SCTLR.M == 1 for the context bank.
- The context bank is not a HYPC bank.
- The context bank is not a MONC bank.

The SMMU_CBn_TCR2.SEP field indicates the bit position that must be used for sign-extending the address. If bits above the sign-extension position are nonzero then a level 0 translation fault is generated.

Note

If the value of SMMU_CBn_TCR2.SEP changes for a context bank, software must invalidate any affected TLB entries.

1.6.5 Address Space Identifiers, ASIDs

In the ARM processor architecture:

- The virtual memory map supports global and non-global regions. Typically, a non-global region is process-specific. Each non-global region has an associated Address Space Identifier (ASID).
- In an implementation that supports processor virtualization, a guest OS is identified by a Virtual Memory Identifier (VMID), see *Virtual machine identifiers, VMIDs on page 1-39*.

Note

SMMUv2 and VMSAv8-64 support 16-bit ASIDs. Although it might be necessary to reuse ASIDs when using 16-bit ASIDs, the requirement to do so, and therefore the information in this section, is less likely to apply.

In ARMv7 and VMSAv8-32, *Address Space Identifiers* (ASIDs) are 8-bits. This limitation means that, after allocating 256 ASIDs, software might be required to reuse ASIDs for different, unrelated processes. To support ASID reuse, the SMMU architecture includes an SMMU_CBn_SCTLR.ASIDPNE bit. When the value of this bit is 1, the SMMU can maintain a private ASID space for each VMID. The SMMU architecture does not require these private ASID spaces to be invalidated by some broadcast TLB invalidation operations. A TLB entry that is allocated when SMMU_CBn_SCTLR.ASIDPNE==1 can include an indication that the entry is excluded from broadcast TLB invalidation operations that target its ASID value.

This ASIDPNE indication is only a hint, and an SMMU can ignore it. In particular, if the SMMU implementation does not support the ASIDPNE feature, broadcast TLB invalidate operations ignore this hint and invalidate matching TLB entries when broadcast TLB invalidation is enabled.

The ASIDPNE hint applies only to broadcast TLB maintenance operations, and does not apply to:

- Global TLB entries.
- Broadcast TLB operations that perform TLB invalidation by VMID matching, unless the invalidation operation specifies both a VMID and an ASID that the TLB has cached for that VMID.
- A TLB invalidation operation initiated locally. That is, a TLB invalidation initiated by a write to a register of the SMMU that holds the TLB.

Note

Whether broadcast TLB maintenance operations affect SMMU TLBs depends on the value of the SMMU_sCR0.{PTM, VMIDPNE} bits.

1.6.6 Virtual machine identifiers, VMIDs

For stage 1 followed by stage 2 translation, the same VMID must be used in both context banks. If different VMIDs are used in stage 1 and stage 2, then the result is UNPREDICTABLE. If a transaction is directed to stage 2, it uses the stage 2 VMID.

An SMMU implementation might provide a Secure VMID for Secure translation regimes. Secure VMIDs are entirely independent of the VMIDs defined for Non-secure translation regimes. Such a Secure VMID is not used for:

- TLB matching for client transactions.
- TLB invalidate operations.
- Broadcast TLB maintenance operations.

All aspects of the use of Secure VMIDs are IMPLEMENTATION DEFINED. For example, A VMID provided for Secure translation regimes might be visible on downstream bus transactions and can be used in an IMPLEMENTATION DEFINED manner. Whether such VMIDs are visible during a translation table walk is IMPLEMENTATION DEFINED.

An SMMU implementation that does not support Secure VMIDs ignores such VMIDs, meaning the following fields are ignored for Secure transactions:

- SMMU_SCR2.BPVMID.
- SMMU_S2CRn.VMID, Bypass mode.
- SMMU_CBARn.VMID, stage 1 context with stage 2 bypass.
- SMMU_CBA2Rn.VMID16, stage 1 context with stage 2 bypass.

Note

The ARM processor architecture does not support Secure VMIDs, and ARM SMMU implementations have no support for Secure VMIDs.

VMID size

SMMUv2 supports either 8-bit VMIDs or 16-bit VMIDs. SMMU_IDR2.VMID16S indicates the supported VMID size, as follows:

VMID16S == 0

Only 8-bit VMIDs are supported.

VMID16S == 1

16-bit VMIDs are supported.

Use of 16-bit VMIDs is enabled by SMMU_CR0.VMID16EN. When enabled the full 16-bit VMID comes from SMMU_CBA2Rn.VMID16 and SMMU_CBARn.VMID becomes RES0. When disabled SMMU_CBA2Rn.VMID16 becomes RES0 and SMMU_CBARn.VMID contains the lower eight bits of the VMID, and the top eight bits of the VMID are zero.

For simplicity, all statements in the architecture that indicate that the VMID comes from SMMU_CBARn.VMID should be read to imply, when 16-bit VMIDs are enabled, that the VMID comes from SMMU_CBA2Rn.VMID16.

Note

When enabled by SMMU_CR0.VMID16EN, 16-bit VMIDs can be used regardless of the value of SMMU_CBA2Rn.VA64.

Changing the value of SMMU_CR0.VMID16EN makes VMIDs of the previous size UNKNOWN. That is:

- If SMMU_CR0.VMID16EN is changed from 0 to 1, the change makes all the 8-bit VMIDs held in SMMU_CBARn.VMID fields UNKNOWN.
- If SMMU_CR0.VMID16EN is changed from 1 to 0, the change makes all the 16-bit VMIDs held in SMMU_CBA2Rn.VMID16 fields UNKNOWN.

Chapter 2
SMMU Operation

This chapter describes the steps that the SMMU performs on receiving a memory access request. It contains the following sections:

Note

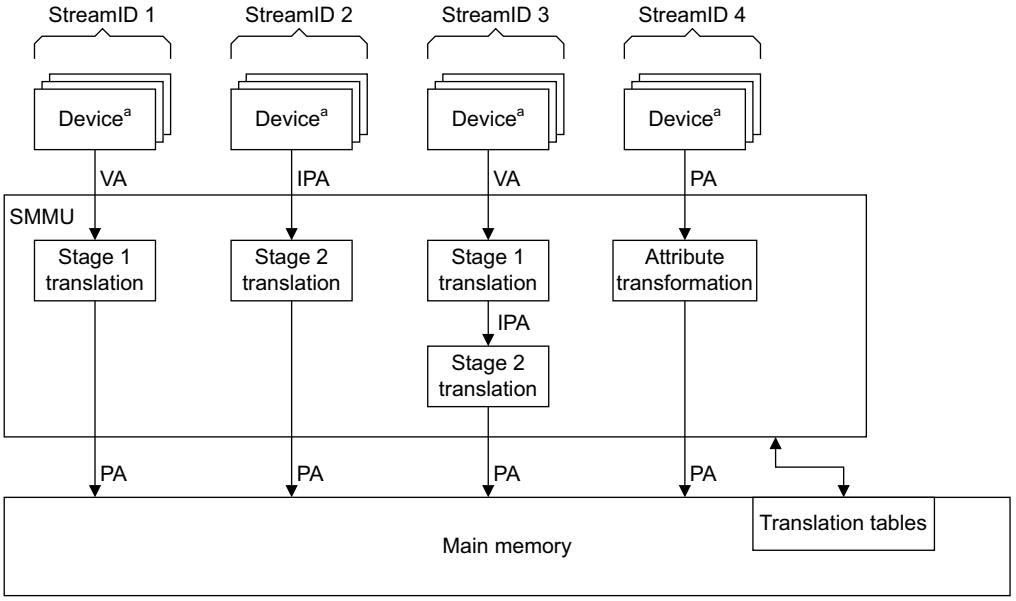
This specification uses a register name scheme described in *Register names on page xiii*. An understanding of this scheme is essential for the correct interpretation of register names.

2.1 Overview of SMMU operation

At the memory system level, in performing address translation, an SMMU controls:

- Security state determination. See *Security State Determination (SSD) on page 2-50*.
- Context determination. See *Context determination on page 2-52*.
- Memory access permissions and determination of memory attributes. See *Memory type and shareability attribute determination on page 2-59*.
- Memory attribute checks.
- TLB operation. See *TLB operation on page 2-72*.

Figure 2-1 shows the SMMU partitioning and the different SMMU transactions.



a. Devices can operate in Secure or Non-secure state.

Figure 2-1 SMMU partitioning

The SMMU architecture supports the use of TLBs to accelerate the translation process, and provides TLB maintenance operations to manage the TLBs.

Security state determination identifies whether a transaction is from a Secure or Non-secure device.

Context determination identifies the *stage 1* or *stage 2 context* resources the SMMU uses to process a transaction. In some cases, the configuration settings for a transaction mean that transaction *bypasses* the translation process, or is *faulted*.

A StreamID uniquely identifies a stream of transactions that can originate from different devices, but are associated with the same context, and are therefore subject to the same type of SMMU processing. The mechanism by which the StreamID is obtained from the incoming bus transaction is IMPLEMENTATION DEFINED. See *Context determination on page 2-52* for more information.

In addition to address translation, the SMMU can modify the memory attributes that a transaction specifies. In some cases, the SMMU performs attribute transformation only, with no address translation. This is shown for StreamID 4 in Figure 2-1, where the input address is the same as the output address. See *Stream-to-Context Register, SMMU_S2CRn on page 2-55* for more information.

Additionally, software can access SMMU address translation registers to initiate address translation operations, where the SMMU returns address information to software. Software-initiated address translation operations are OPTIONAL in SMMUv2. See Chapter 4 *Address Translation Operations* for more information.

An access to the SMMU is referred to as a *transaction*:

- A *client transaction* is an access by a client device, that the SMMU is to process.
- A *configuration transaction* is a device access to a register in the SMMU configuration address space.

Figure 2-2 on page 2-44 shows the generic SMMU process flow, described in the remaining sections of this chapter.

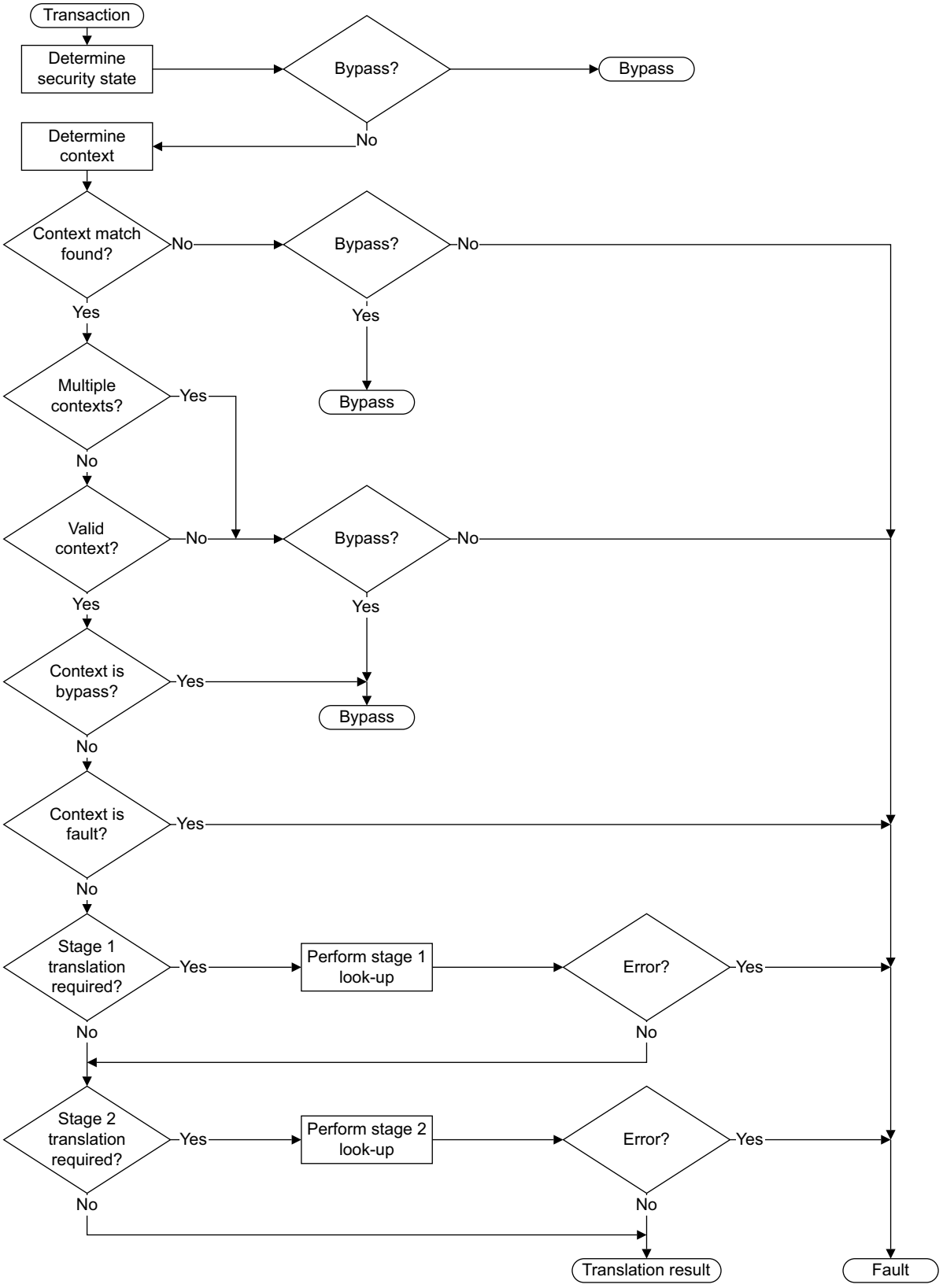


Figure 2-2 SMMU process flow

Note

For a transaction that requires two stages of address translation, as described in *About the ARM System MMU architecture on page 1-18*, addresses for the stage 1 translation are defined in the IPA address space. In this case, for each memory access performed as part of a stage 1 translation table walk the SMMU must perform a stage 2 translation of the address, to map the IPA to the corresponding PA. This applies both to addresses held in the stage 1 *Translation Table Base Register* (TTBR) and those obtained from table entries in the stage 1 translation tables. This stage 2 translation might fail, generating an error. Figure 2-2 on page 2-44 does not show this possible dependence of a stage 1 lookup on a stage 2 translation.

2.1.1 SMMU transaction processing

This section describes how the SMMU processes transactions. It provides an overview of context determination, and information about the registers it uses to perform each operation.

Figure 2-3 on page 2-46 shows an example of the SMMU processing a transaction. IMPLEMENTATION DEFINED choices for the SMMU implementation mean that other processing flows are possible. For example, StreamID indexing can replace StreamID matching.

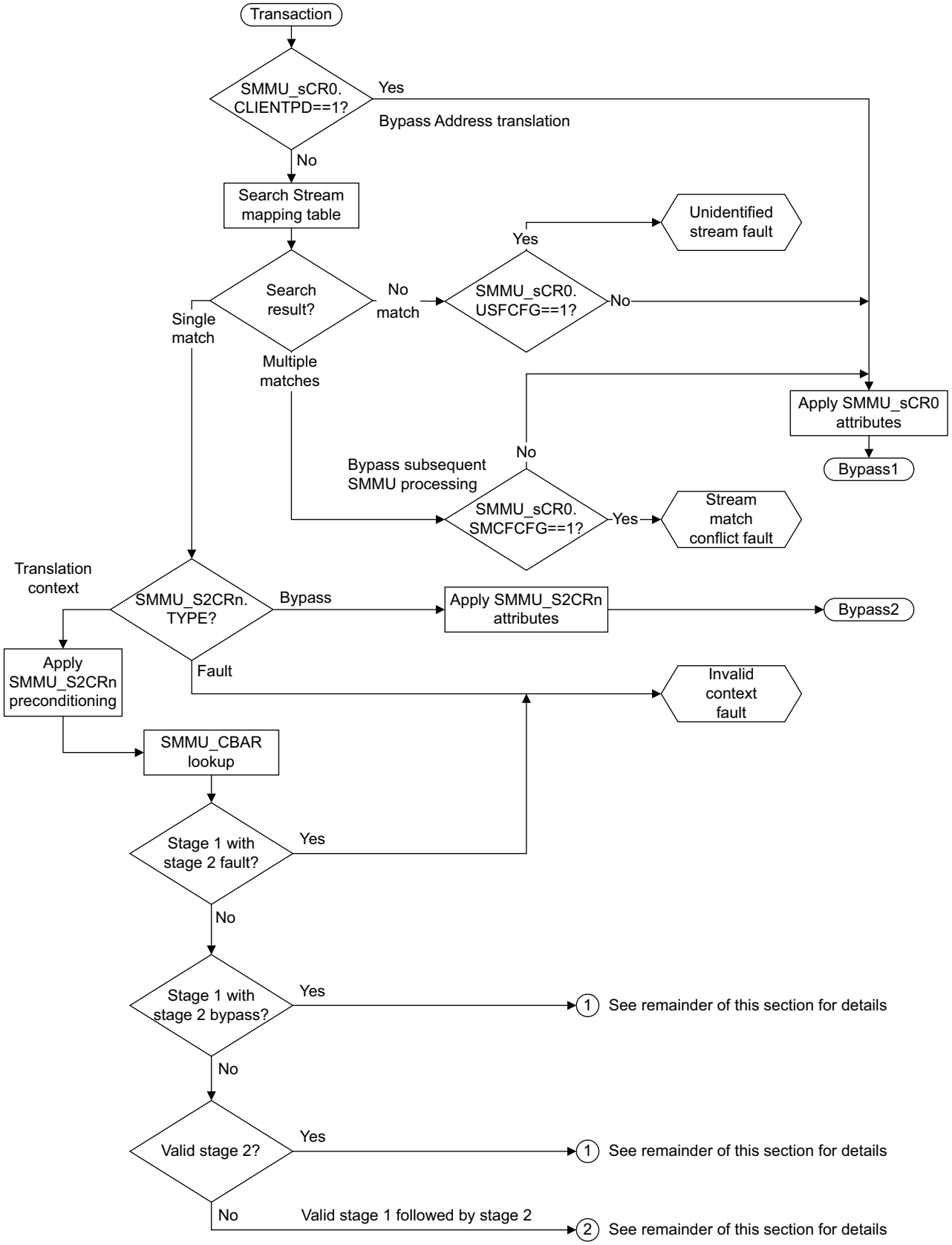


Figure 2-3 Example SMMU transaction processing, using Stream matching

If SMMU_sCR0.CLIENTPD==1, the transaction bypasses:

- SMMU translation.
- Protection checking. However, a Secure instruction fetch is subject to a protection check based on the value of SMMU_SCR1.SIF, regardless of the value of SMMU_SCR0.CLIENTPD.
- Attribute generation.

When a transaction bypasses address translation, due to SMMU_SCR0.CLIENTPD==1:

- The address is not translated, meaning that the output address is identical to the input address.
- Output attributes of the transaction are a function of the input attributes and SMMU_sCR0 fields.

When using Stream matching, if the value of SMMU_sCR0.CLIENTPD is 0, indicating that the transaction is to be processed, the SMMU searches the Stream mapping table for a matching Stream mapping register group. See *The Stream mapping table on page 2-54*.

If no match is found, SMMU_sCR0.USFCFG determines how to handle the match failure. If the bypass action is specified, the address is not translated, and the output attributes are a function of the input attributes and SMMU_sCR0. Otherwise, an Unidentified stream fault is generated.

If a search of the Stream mapping table yields multiple matches, SMMU_sCR0.SMCFCFG determines whether the transaction bypasses subsequent SMMU processing or generates a Stream match conflict fault.

If the transaction is successfully matched in the Stream mapping table, SMMU_S2CRn determines the initial context. If SMMU_S2CRn specifies Bypass mode:

- The address is not translated, meaning that the output address is identical to the input address.
- Output attributes are a function of the input attributes and the SMMU_S2CRn fields.

If SMMU_S2CRn specifies that the initial context is a translation context bank, the input attributes to the translation process are those specified by the transaction, unless the SMMU_S2CRn fields specify replacement attributes. SMMU_CBARn defines additional configuration for the translation context bank.

Processing a Stage 1 with stage 2 bypass or a Stage 2 translation context or *Processing a Stage 1 followed by stage 2 context on page 2-49* describes the processing of the transaction.

Processing a Stage 1 with stage 2 bypass or a Stage 2 translation context

Figure 2-4 shows the processing of a Stage 1 with Stage 2 bypass context, or a Stage 2 context.

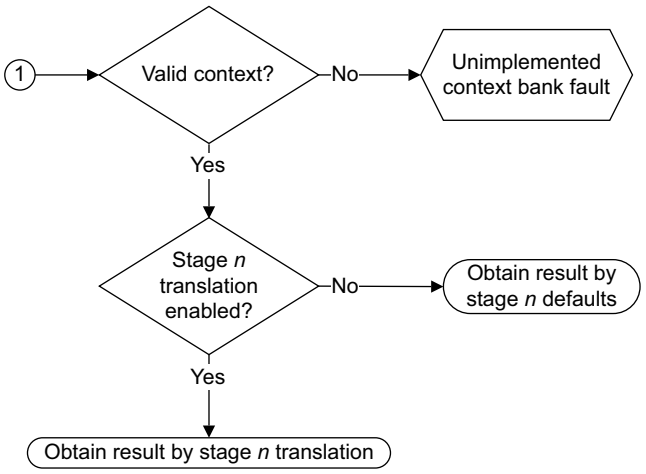


Figure 2-4 Stage 1 with stage 2 bypass, or stage 2 translation context

For a stage 1 with stage 2 bypass translation context:

- If the stage 1 context is not valid, an Unimplemented context bank fault is generated and there is no further processing of this transaction.
- The stage 1 SMMU_CBn_SCTLR.M is read to see if stage 1 translation is enabled, and:
  - If the value of SMMU_CBn_SCTLR.M is 1, indicating that stage 1 address translation is enabled, the stage 1 translation lookup is performed, using the information in the context bankSMMU_CBARn.
  - If the value of SMMU_CBn_SCTLR.M is 0, indicating that stage 1 address translation is disabled, the output values for this stage are obtained from the input attributes, SMMU_S2CRn, and the stage 1 SMMU_CBn_SCTLR, SMMU_CBARn and SMMU_S2CRn.

The stage 1 output attributes are combined with a stage 2 only attribute transformation contained in the stage 1 SMMU_CBARn.

For a stage 2 only translation context:

- If the stage 2 context is not valid, an Unimplemented context bank fault is generated and there is no further processing of this transaction.
- The stage 2 SMMU_CBn_SCTLR.M is read to see if stage 2 translation is enabled, and:
  - If the value of SMMU_CBn_SCTLR.M is 1, indicating that stage 2 address translation is enabled, the stage 2 translation lookup is performed.
  - If the value of SMMU_CBn_SCTLR.M is 0, indicating that stage 2 address translation is disabled, the output values for this stage are obtained from the stage 2 SMMU_CBn_SCTLR.

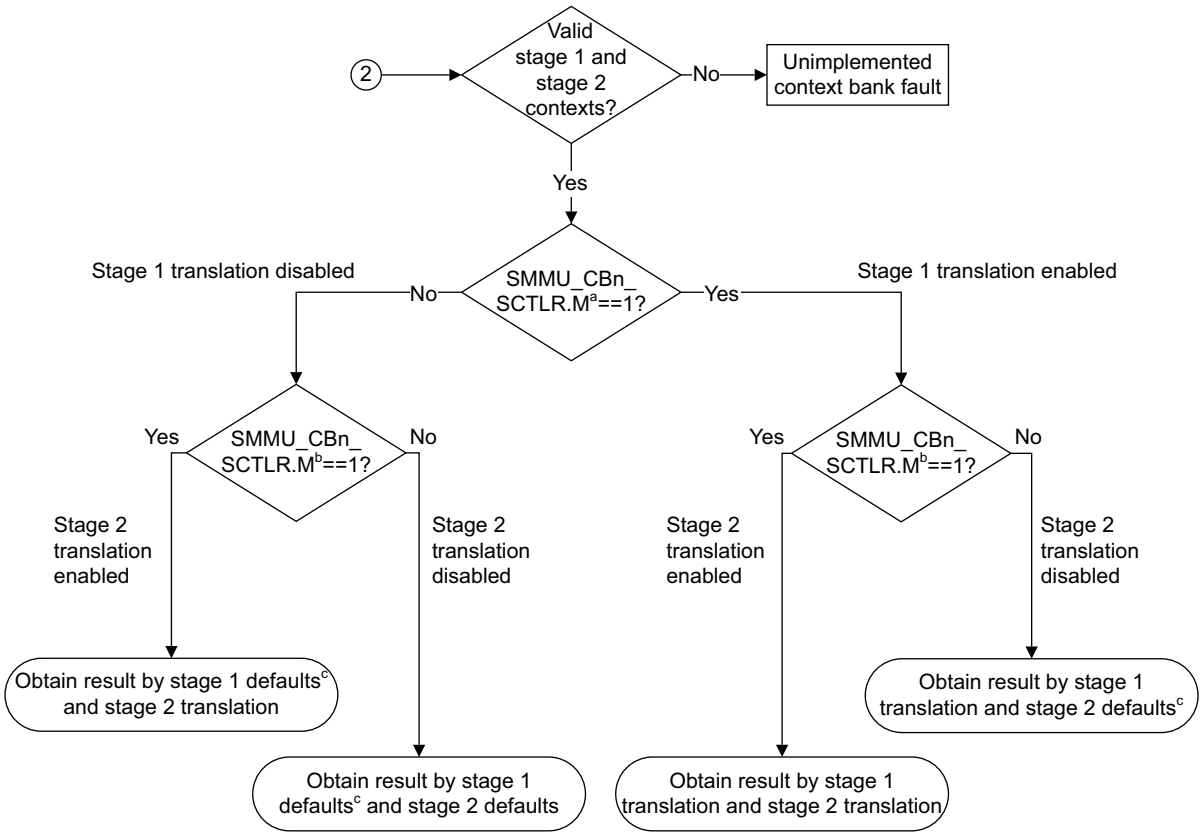
The input attributes that have been modified by SMMU_S2CRn form a notional stage 1 attribute and are then combined with the output values of the stage 2.

Note

When a translation lookup is performed, the result is obtained from the TLB, if possible. If the required translation is not cached in the TLB, the result is obtained by performing a translation table walk.

Processing a Stage 1 followed by stage 2 context

Figure 2-5 shows the processing of a Stage 1 followed by stage 2 transaction.



a. Stage 1 translation context bank register.
b. Stage 2 translation context bank register.
c. Obtained from corresponding SMMU_CBn_SCTLR.

Figure 2-5 Stage 1 followed by stage 2 translation context

For stage 1 followed by stage 2 translation context:

- The stage 1 SMMU_CBn_SCTLR.M bit controls whether the stage 1 translation is enabled, and the stage 2 SMMU_CBn_SCTLR.M bit controls whether stage 2 translation is enabled.
- If the stage 1 and stage 2 contexts are valid, but stage 1 translation is not enabled:
  - If stage 2 translation is enabled, the result is obtained from the default input attributes, SMMU_S2CRn, the stage 1 SMMU_CBn_SCTLR fields, and a stage 2 translation table lookup
  - If stage 2 translation is not enabled, the result is obtained from the default input attributes, SMMU_S2CRn, the stage 1 SMMU_CBn_SCTLR fields, and the stage 2 SMMU_CBn_SCTLR fields.
- If the stage 1 and stage 2 contexts are valid, and stage 1 translation is enabled:
  - If stage 2 translation is enabled, the result is obtained from the stage 1 translation table lookup, and the stage 2 translation table lookup
  - If stage 2 translation is not enabled, the result is obtained from the stage 1 translation table lookup and the stage 2 SMMU_CBn_SCTLR fields.

2.2 Security State Determination (SSD)

The SMMU architecture supports shared use of an SMMU by devices executing in Secure state and devices executing in Non-secure state:

- A device executing in Secure state can request translations that result in Secure or Non-secure downstream transactions.
- A device executing in Non-secure state can request translations that result in Non-secure downstream transactions.

The first step of transaction processing identifies which Security state a transaction belongs to, and therefore the permitted behaviors of the transaction. This step is called *Security State Determination* (SSD).

Note

SMMU support for two Security states is OPTIONAL. See Chapter 7 *SMMU Support for Two Security States* for more information.

In an SMMU implementation that supports two Security states:

- A device can transition between Secure and Non-secure state dynamically.

  Note

  Only Secure software can transition a device between Security states.

- A Secure device might be able to produce both Secure and Non-secure bus-marked transactions.
- There is no guarantee that a device that is in Non-secure state cannot mark a transaction as Secure.

For these reasons, the bus-marked Security state of a transaction arriving at the SMMU might be insufficient to determine whether a memory access originated from a Secure or a Non-secure device. The SMMU architecture therefore defines the *SSD security state*, that indicates whether a particular device or stream is in the Secure or the Non-secure domain at the time it initiates a transaction. For a given transaction, the upstream device is either SSD Secure or SSD Non-secure.

An upstream SSD Secure device can access both Non-secure and Secure downstream devices and memory. An upstream SSD Non-secure device can access only Non-secure downstream devices and memory,

The SSD Security state is independent of any upstream security marking that the underlying bus system might apply to a transaction. This Specification refers to Secure bus markings or Non-secure bus markings in cases where a bus system applies a Security state attribute to a transaction. For example, an SSD Secure device might issue a Non-secure transaction, in which case:

- The SSD security state is Secure.
- The transaction has a Non-secure bus marking.

Each stream has a single SSD Security state. Where several devices issue transactions as part of a single stream, these transactions must all have the same SSD Security state.

A single device might also issue transactions to multiple streams, which might have different SSD Security states.

The SMMU architecture defines the following rules for the SSD Security state. Where the incoming transaction is:

- SSD Non-secure, the SMMU cannot generate downstream transactions with a Secure bus marking.
- SSD Secure, the SMMU can generate downstream transactions with a Secure or a Non-secure bus marking.

The mechanism by which a system determines the SSD Security state depends on whether the SSD address space is present, as defined by the SMMU_IDR1 SSDTP bit. When this address space is present, SMMU_SSDR registers are provided in the SMMU address space.

SSD address space is present

An IMPLEMENTATION DEFINED mechanism associates each transaction with an SSD_Index. The SSD_Index is used to select a bit in the SMMU_SSDRn register, the value of which determines the SSD Security state of the transaction. See *SMMU SSD address space on page 13-272* for more information.

Before transferring control to Non-secure software, to ensure correct operation Secure software must access the appropriate SMMU_SSDRn register and:

1. Set to 0 all bits corresponding to devices that must be Secure.
2. Set to 1 all other bits.
3. Read all bits to verify correct operation, ensuring that any IMPLEMENTATION DEFINED bits are set correctly.

SSD address space is not present

An SMMU implementation and the system that incorporates it can adopt alternative IMPLEMENTATION DEFINED approaches to determine the Security state of a transaction. For example:

- The transaction source can determine a Security state and propagate this with the transaction to the SMMU. For example, an SSD Secure device might issue Secure and Non-secure transactions on the upstream bus and provide an additional signal indicating that the transaction belongs to an SSD Secure device.
- If an SSD Secure device can issue only Secure transactions to the SMMU, and no SSD Non-secure device can issue Secure transactions to the SMMU, an SMMU implementation can obtain the SSD Security state from the transaction security signaling on the upstream bus from which it receives those transactions.
- The SMMU can use other platform-specific knowledge to determine the SSD security status of each transaction.

In either case, to prevent any device incorrectly being configured as SSD Secure, Secure software must:

- Be aware of the Security state determination mechanism.
- Be aware of all devices that can issue transactions to the SMMU and can operate as Secure.
- Be able to identify the SSD Security state determination mechanism it must use with the system.

2.2.1 Banked registers

Some SMMU registers are banked by security. This means that a Non-secure access to a register address accesses the Non-secure copy of the register, and a Secure access accesses the Secure copy in the Secure address space, at the same address offset as its Non-secure counterpart.

Not all registers available in both the Secure and Non-secure states are Banked. For example, registers in translation context banks that are allocated for Secure or Non-secure access are not Banked. All registers in Secure translation context banks are invisible to Non-secure accesses.

For information about banked registers and naming conventions, see *Register names on page xiii*.

Additional alias registers are provided at different offset addresses, permitting Secure software to access Non-secure versions of the resources. See *Secure alias for Non-secure registers on page 9-177* for more information.

2.3 Context determination

The SMMU processes a transaction in one of the following ways:

- Bypass address translation but transform attributes.
- Fault the transaction.
- Require the resources of either one or two translation context banks, each having its own set of translations, attributes and permissions that apply to a transaction being processed by that context bank.

Context determination determines the resources the SMMU uses to process a transaction.

In an SMMU that supports two stages of translation, a transaction can be associated with up to two translation context banks.

2.3.1 Transaction streams

The SMMU can process transactions from multiple sources, potentially using a different context for each transaction. A *transaction stream* is a sequence of transactions associated with a particular thread of activity in the system.

Transactions from the same transaction stream are associated with the same context, and are therefore subject to the same type of processing in the SMMU. A device in the system can issue transactions using more than one transaction stream, and a single transaction stream can contain transactions from more than one device.

A *Stream Identifier* (StreamID) associates a transaction with a transaction stream. The StreamID is derived from transaction identification information, such as:

- The transaction ID.
- The read and write status.
- The Security state of the upstream bus transaction.

As a result of the system-specific nature of transaction identification, and the variety of system interconnect protocols that exist, the encoding of the StreamID used by an SMMU implementation is IMPLEMENTATION DEFINED.

Note

The StreamID uniquely identifies a stream of transactions, and can commonly be derived from identifier information conveyed on the bus interconnect, such as the NS bit, the *Read not Write* (RnW) bit indicating whether a transaction is a read or a write operation, and the transaction ID.

StreamID size

The number of implemented StreamID bits:

- Is defined by SMMU_IDR0.NUMSIDB[3:0], with possible extension to 16 bits in SMMUv2.
- Lies in the range:
  - 0-15, in SMMUv1.
  - 0-16, in SMMUv2.

In SMMUv2, support for 16-bit StreamIDs is optional. 16-bit Stream IDs are supported when both:

- SMMU_IDR0.EXIDS==1.
- SMMU_IDR0.NUMSIDB==15.

When SMMU_sCR0.EXIDENABLE==1, 16-bit StreamID functionality is enabled, and if stream matching is used the stream matching process uses the extended ID SMMU_SMRn register format.

A zero-bit StreamID might exist if the source of a single StreamID has a dedicated SMMU.

2.3.2 Stream mapping

Stream mapping is the process of mapping a StreamID to the associated *Stream-to-Context* register, SMMU_S2CRn, that defines the processing required for the stream. SMMUv2 defines three stream mapping mechanisms:

Stream matching

The StreamID is looked up in the set of *Stream Match* registers, SMMU_SMRn. When a unique match is found, the corresponding SMMU_S2CRn holds the context for the stream.

SMMUv1 provides up to 128 Stream Match registers. An SMMUv2 implementation can provide up to 128 Stream Match Registers, or can implement the optional Extended Stream Matching Extension, that provides up to 1024 Stream Match Registers and associated Stream-to-Context registers. Chapter 14 *Extended Stream Matching Extension* describes this extension.

Note

In an implementation that includes the Extended Stream Matching Extension, the Stream Match registers that are present if the extension is absent or disabled map onto the final 128 Stream Match registers provided by the extension.

Stream indexing

When Stream indexing is used, the StreamID is a direct index to the required SMMU_S2CRn. That is, if the StreamID is *m*, the required Stream-to-Context register is SMMU_S2CR*m*. This means the maximum StreamID is determined by the number of implemented SMMU_S2CRns, and is:

- Up to 127 in an SMMU implementation that does not include the Extended Stream Matching Extension.

Compressed StreamID indexing

Compressed stream indexing is available only in an SMMUv2 implementation that includes the optional StreamID Compressed Indexing extension. Chapter 15 *StreamID Compressed Indexing Extension* describes this extension.

When Compressed StreamID indexing is used, the StreamID is an indirect index to the required SMMU_S2CRn, as follows:

1. The StreamID, *m*, indexes a single-byte S2CRIndex*i* field in the array of SMMU_COMPINDEXn registers provided by the StreamID Compressed Indexing extension.
2. The S2CRIndex*i* field holds the value of the SMMU_S2CRn for the stream. So, if the value of the S2CRIndex*i* field is *m1*, the required Stream-to-Context register is SMMU_S2CR*m1*.

   Note

   A stream is always identified by its StreamID, not by the value of the corresponding S2CRIndex*i* field.

Only a single Security state can use Compressed StreamID indexing. In an SMMUv2 implementation that supports two Security states, if one Security state is using Compressed StreamID indexing then the other Security state must use Stream matching.

Regardless of the stream mapping mechanism being used, the StreamID size is defined as described in *StreamID size on page 2-52*. However, when Stream indexing is used, the maximum StreamID might be constrained by the number of implemented SMMU_S2CRns, as described in *Stream indexing*.

Stream-to-Context Register, SMMU_S2CRn on page 2-55 gives more information about the context defined by the SMMU_S2CRns.

Permitted stream mapping implementation options

Only certain combinations of stream mapping options are permitted. Table 2-1 shows the permitted options:

Table 2-1 Permitted combinations of stream mapping mechanisms

Stream matching or indexing	Extended stream matching	Compressed StreamID indexing	Notes
Matching	No	No	Up to 128 Stream Match register groups.
		Yes[a]	Up to 128 SMMU_S2CRn registers. Each StreamID can be directly assigned to an SMMU_S2CRn.
	Yes	No	Up to 1024 Stream Match register groups.
		Yes[a]	Up to 1024 Stream Match register groups. Compressed indexing can make use of only the last 128 of these.
Indexing	No	No	Up to 128 SMMU_S2CRn. The maximum StreamID size is 7 bits. The size is constrained by the number of SMMU_S2CRn registers implemented.

a. When Compressed StreamID indexing is supported, it can be used in only one Security state.

Support for the different stream mapping options is indicated as follows:

- SMMU_IDR0.SMS indicates whether an implementation supports Stream matching or Stream indexing. A value of 1 indicates support for Stream matching.
- SMMU_IDR0.EXSMRGS indicates whether an implementation supports Extended stream matching. A value of 1 indicates support for Extended stream matching.
- SMMU_IDR2.COMPINDEXS indicates whether an implementation supports Compressed StreamID indexing, and is 1 if this is supported. In an implementation that supports two Security states, a Secure read of SMMU_IDR2 is needed to determine this support. A Non-secure read indicates only whether Compressed StreamID indexing is available for Non-secure transactions.

2.3.3 The Stream mapping table

The Stream mapping table maps a transaction stream to a context. It consists of a number of entries, where each entry is a Stream mapping register group containing the following registers, with n defining the Stream mapping register group number:

- SMMU_SMRn
- SMMU_S2CRn.

Note

In an implementation that does not support Stream matching it is still possible to refer to the Stream mapping table, but the SMMU_SMRns are RES0.

In an implementation that uses StreamID matching, SMMU_SMRn determines whether a transaction matches the group, and if so SMMU_S2CRn specifies the initial context for the translation process, as described in *Stream-to-Context Register, SMMU_S2CRn on page 2-55*.

SMMU_IDR0.NUMSMRG specifies the number of IMPLEMENTATION DEFINED Stream mapping register groups. In an implementation that supports two Security states, Secure and Non-secure register groups can be implemented. Secure software can access both Secure and Non-secure register groups. An attempt by Non-secure software to access a Secure group, or by any software to access a group above the reported implemented number, results in one of the following IMPLEMENTATION DEFINED behaviors:

- The access is treated as RAZ/WI.

- A configuration access fault is raised, as Chapter 3 *The Fault Model* describes.

In an implementation that supports Stream matching, ARM recommends that Secure software gives Non-secure software at least one Stream Match register group.

2.3.4 Stream-to-Context Register, SMMU_S2CR*n*

SMMU_S2CRn specifies the initial context for processing a transaction. Optionally, SMMU_S2CRn also preconditions the incoming attributes. This preconditioning stage can alter incorrect attributes from a client device, or can override the incoming permission attributes as required for correct interaction in the translation tables.

n is in the range 0-127, and identifies the Stream mapping register group for the transaction. This means that the other register in the Stream mapping register group is SMMU_SMRn.

SMMU_S2CRn specifies one of the following initial contexts:

- A translation context
  - A stage 1 translation context bank, if the implementation supports stage 1 translation.
  - A stage 2 translation context bank, if the implementation supports stage 2 translation.
- Bypass mode, optionally overlaying memory attributes
- Fault.

Translation context bank

The SMMU includes separate stage 1 and stage 2 translation context banks that provide functionality for stage 1 and stage 2 translation, such as translation process configuration and TLB maintenance operations.

An implementation can support the following types of translation:

- Stage 1 only.
- Stage 2 only.
- Stage 1 followed by stage 2.

If SMMU_S2CRn specifies a stage 1 translation context bank or a stage 2 translation context bank as the initial context, the SMMU_CBARn register associated with the translation context bank can specify additional translation attributes.

Additionally, in an implementation that supports stage 1 followed by stage 2 translation, when SMMU_S2CRn specifies a stage 1 translation context bank, the SMMU_CBARn register associated with the initial translation context bank can specify a subsequent stage 2 context bank.

The number of SMMU_CBARn registers matches the number of entries in the translation context bank table, and is IMPLEMENTATION DEFINED. SMMU_IDR1.NUMCB specifies the number of implemented SMMU_CBARn registers.

For more information about attributes associated with a translation context bank, see *The Context Bank Attribute Register, SMMU_CBARn on page 2-78*.

SMMU_S2CRn can specify:

- The following memory attributes to replace those specified by the transaction:
  - Memory type, using the MemAttr field when MTCFG is set to 1.
  - Shareable attributes, using the SHCFG field.
  - Cache allocation hints, using the WACFG and RACFG fields.
  - For Secure Stream mapping register groups only, Non-secure configuration, using the NSCFG field.
- The following permission attributes that indicate device privilege level:
  - Instruction fetch attributes, using the INSTCFG field.
  - Privilege attribute, using the PRIVCFG field.

These attributes are the starting point for translation. The translation context bank can modify them. See *Memory type and shareability attribute determination on page 2-59* for more information.

It is IMPLEMENTATION DEFINED whether an SMMU implementation supports:

- Stage 1 translation. SMMU_IDR0.S1TS specifies whether stage 1 translation is supported.
- Stage 2 translation. SMMU_IDR0.S2TS specifies whether stage 2 translation is supported.
- Stage 1 followed by stage 2 translation. SMMU_IDR0.NTS specifies whether stage 1 followed by stage 2 translation is supported.

Note

If stage 1 followed by stage 2 translation is supported, then both stage 1 and stage 2 translation must be supported. That is, if the SMMU_IDR0.NTS bit is set to 1, then both the SMMU_IDR0.S1TS and SMMU_IDR0.S2TS bits must be set to 1.

The number of translation context bank entries for stage 1 or stage 2 translation is IMPLEMENTATION DEFINED. If SMMU_S2CRn is configured to specify an unimplemented translation context bank for stage 1 or stage 2, any transaction processed as part of that stream results in an Unimplemented context bank fault.

Bypass mode

If SMMU_S2CRn.TYPE is configured as Bypass mode, no address translation is applied to the transaction. However, SMMU_S2CRn can specify memory and permission attributes for the transaction.

No protection check is applied, except where an instruction fetch by a Secure client is subject to a protection check, based on the value of SMMU_SCR1.SIF.

Fault

If SMMU_S2CRn specifies a fault, all transactions associated with that Stream mapping register group are subject to an invalid context fault. See *Global faults on page 3-114* for more information.

2.3.5 Bypassing the Stream mapping table

A transaction bypasses the Stream mapping table if any of the following apply:

- SMMU_sCR0.CLIENTPD is 1.
- The transaction does not match any entries in the Stream mapping table and SMMU_sCR0.USFCFG is 0.
- The transaction is detected to match multiple entries in the Stream mapping table and SMMU_sCR0.SMCFCFG is 0.

A transaction that bypasses the Stream mapping table is not subject to address translation, but is subject to an attribute-only transformation from the appropriate copy of SMMU_sCR0. In most cases, such a transaction is not subject to any protection checking. In particular:

- A data access is not subject to any protection checking.
- An instruction fetch might be subject to a permission check if the implementation supports two Security states.

In an implementation that supports two Security states:

- SMMU_CR0.CLIENTPD only applies to Non-secure transactions
- SMMU_SCR0.CLIENTPD is an equivalent field that applies to Secure transactions.

Secure instruction fetch transactions that bypass the Stream mapping table can be subject to a protection check. If the SMMU_SCR1.SIF bit is set to 1, a permission fault is recorded for any incoming SSD Secure transaction where the SMMU generates a downstream transaction with a Non-secure instruction bus marking.

This check applies globally, and records faults to SMMU_SGFSR. If a transaction is associated with a particular translation context bank, faults are recorded in SMMU_CBn_FSR instead of SMMU_SGFSR.

Instruction fetches by a Non-secure client are not subject to any protection checking.

Note

Whether Secure Instruction Fetch checks apply to instruction writes is IMPLEMENTATION DEFINED.

2.3.6 No match handling

If no match for a transaction is found in the Stream mapping table, the SMMU can either:

- Permit the transaction to bypass the SMMU translation process.
- Fault the transaction, resulting in an unidentified stream fault.

SMMU_sCR0.USFCFG can be used to configure the handling of a transaction that does not match any Stream mapping table entry. In an implementation that supports two Security states:

- SMMU_CR0.USFCFG only applies to Non-secure transactions.
- SMMU_SCR0.USFCFG applies to Secure transactions.

Transactions that bypass the SMMU translation process are not subject to address translation. SMMU_sCR0 can be used to configure a set of bypass memory attributes that apply to all such transactions.

2.3.7 StreamID matching

In a register group that uses StreamID matching, SMMU_SMRn specifies conditions that must be met for a transaction to be associated with the Stream mapping register group to which SMMU_SMRn belongs.

The SMMU_SMRn registers form a table that is searched associatively to find a match for the StreamID of a transaction. For more information about StreamIDs, see *Transaction streams on page 2-52*.

If a transaction matches all the conditions specified in SMMU_SMRn, the translation context, Bypass mode, or fault context specified by SMMU_S2CRn is used to process the transaction.

SMMU_SMRn provides StreamID and mask fields that permit the masking of StreamID bits irrelevant to the matching process.

The Stream mapping table permits a number of StreamID values to be mapped to the same context. This means the state describing that context can be shared. Mapping multiple StreamID values to the same context is achieved using multiple Stream mapping table entries, or using the mask facilities in the SMMU_SMRn encoding.

The Stream mapping table must be configured so that a StreamID matches, at most, one entry in the Stream mapping table. During configuration, software must ensure that there is no overlap in Stream mapping table entries for all StreamIDs that are active.

If the StreamID of a transaction matches multiple Stream mapping table entries, either of the following IMPLEMENTATION DEFINED options are possible:

- The SMMU detects the multiple match and either:
  - Permits the transaction to bypass the SMMU translation process.
  - Faults the transaction with a Stream match conflict fault.
- The SMMU does not detect the multiple match, and processes the transaction using one of the matching entries in the Stream mapping table.

SMMU_IDR1.SMCD indicates whether the SMMU detects all Stream match conflicts. This value is IMPLEMENTATION DEFINED.

SMMU_sCR0.SMCFCFG specifies whether the SMMU permits bypass or raises a Stream match conflict fault. It is IMPLEMENTATION DEFINED whether this setting is configurable. If it is not configurable, SMMU_sCR0.SMCFCFG has a fixed value and writes are ignored.

SMMU_sCR0.USFCFG specifies how transactions with no match in the Stream mapping table are handled. It is IMPLEMENTATION DEFINED whether the SMMU supports all the possible handling options.

Unimplemented SMMU_SMRn registers, or those allocated for use by Secure software and therefore invisible to Non-secure accesses, behave as RAZ/WI. These registers have an IMPLEMENTATION DEFINED option to trap accesses to unimplemented registers and raise a configuration access fault. See Chapter 3 *The Fault Model*.

2.3.8 StreamID indexing

In an implementation that uses StreamID indexing, a one-to-one stream mapping associates a transaction with a Stream mapping register group. The StreamID associated with the transaction is an index into the Stream mapping table, and directly selects the Stream mapping register group for processing the transaction. The maximum StreamID size in a register group that uses StreamID indexing is 7 bits.

It is IMPLEMENTATION DEFINED whether the Stream mapping table implements Stream Match Register functionality. SMMU_IDR0.SMS indicates whether the register group supports Stream Match Register functionality. In an implementation that supports StreamID indexing, a Stream Match Register has no effect on the stream mapping process and is UNK/SBPZ.

2.3.9 The Stream mapping table in an implementation that supports two Security states

In an implementation that supports two Security states, the Stream mapping table is a common resource. Secure software can reduce the size of the Stream mapping table seen by Non-secure software using SMMU_SCR1.NSNUMSMRGO. Non-secure accesses see only the table size defined by SMMU_SCR1.NSNUMSMRGO, and this value determines the size of the table seen by a Non-secure read of SMMU_IDR0.NUMSMRG. This functionality permits Secure software to allocate a number of Stream mapping register groups to be used by Secure software.

Secure software can access all implemented entries in the table. This means Secure software can inspect Non-secure Stream mapping table entries.

The stream mapping process uses the Secure status indication from the Security state determination to prevent Non-secure access, as follows:

- Transactions from Non-secure devices can only match Stream mapping table entries not allocated for use by Secure software.
- Transactions from Secure devices can only match Stream mapping table entries allocated for use by Secure software.

Where using SMMU_SCR1.NSNUMSMRGO to allocate the SMMU_S2CRn registers for Secure use, SMMU_S2CRn must specify a translation context bank of appropriate security, as follows:

- An SMMU_S2CRn register allocated for use by Secure software can only specify a translation context bank that is allocated for use by Secure software. If Secure software fails to comply with this requirement, the result is UNPREDICTABLE.
- An SMMU_S2CRn register not allocated for use by Secure software can only specify a translation context bank that is not allocated for use by Secure software. If Non-secure software fails to comply with this requirement, an Unimplemented context bank fault is raised for any transactions mapped to that SMMU_S2CRn register.

2.3.10 Reset state

On reset, no initialization is performed on Stream mapping table entries. The required contents of the Stream mapping table must be initialized before the SMMU is enabled.

2.4 Memory type and shareability attribute determination

A transaction can pass through a number of steps as part of the SMMU translation process. The memory type and shareability attributes can be obtained by combining from a number of sources.

If the final attribute for a memory location is Strongly-ordered or device, then the shareability is treated as Outer Shareable.

In SMMUv2, the SMMU treats final attributes that are Normal Inner Non-cacheable or Normal Outer Non-cacheable as Outer Shareable. In SMMUv1, it is IMPLEMENTATION DEFINED how the SMMU treats such attributes.

See the *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition* for information about shareability for these memory types.

In general, the final attributes used for a transaction depend on fields in several registers, in addition to the contents of the translation table. Table 2-2 shows, for each transaction type, whether each register field applies.

Table 2-2 Location and applicability of final memory and shareability attribute information

Location	NSCFG[a]	WACFG	RACFG	MemAttr	MTCFG	SHCFG	INSTCFG	PRIVCFG
Stage 1 translation table	Y	SMMU_CBn_MAIRm provides memory attribute selection.			N	Y	Partially determine permission fault	
Stage 2 translation table	N	Y[b]	Y[b]	Y	N	Y	Partially determine permission fault	
SMMU_S2CRn, Bypass mode	Y	Y	Y	Y	Y	Y	N	N
SMMU_S2CRn, translation context	Y	Y	Y	Y	Y	Y	Y[c]	Y
Stage 1 SMMU_CBn_SCTLR[d]	Y	Y	Y	Y	Y	Y	N	N
Stage 2 SMMU_CBn_SCTLR[e]	N	Y	Y	Y	Y	Y	N	N
SMMU_CBARn[f]	N	Y	Y	Y	N	Y	N	N

a. This bit exists only for Secure transactions.
b. Read Allocate and Write Allocate are not defined for processor stage 2 translation tables. However, bits are reserved for the SMMU for this purpose See Table 2-12 on page 2-84 for more information.
c. For reads only. In SMMUv2, all writes are forced to `0b10`, indicating data. In SMMUv1, it is IMPLEMENTATION DEFINED whether this applies to writes.
d. When stage 1 context bank MMU behavior is disabled.
e. When stage 2 context bank MMU behavior is disabled.
f. When SMMU_CBARn.TYPE specifies stage 1 with stage 2 bypass, this register specifies the stage 2 attributes that are combined with the stage 1 attributes.

Note

In SMMUv2, when any of the attributes in Table 2-2 are encoded as Reserved, the meanings are as described in *Reserved memory type and shareability attributes on page 2-62*.

Table 2-3 shows how the SMMU must treat the upstream bus marking in cases where the upstream bus system does not support a particular bus marking.

Table 2-3 SMMU treatment of unsupported upstream bus marking

Attribute	Treated as
NSCFG	Non-secure
RACFG WACFG	Allocate
MemAttr	Write-Back, Write-Allocate
SHCFG	Non-shared
INSTCFG	Data
PRIVCFG	Unprivileged
TRANSIENTCFG	Non-transient

If the downstream bus system does not support a particular bus marking, an SMMU implementation might not provide storage for the attribute and, with the exception of NSCFG and PRIVCFG, the corresponding field in Table 2-3 might be RAZ/WI.

Note

- If stage 1 is supported, then NSCFG and PRIVCFG interact with the security and permission models and must be retained.
- If stage 1 is not supported or for stage 2 only context banks, then these may be RAZ/WI because stage 2 does not support Secure transactions and does not have a permission model with two levels of privilege.
- In implementations where upstream devices have no concept of “instruction” then an implementation may choose to make INSTCFG RAZ/WI.

If any SMMU_sCR0 settings result in a transaction bypassing the Stream mapping table, as *Bypassing the Stream mapping table on page 2-56* describes, then SMMU_sCR0 specifies memory attribute transformation for the transaction.

Similarly, if the SMMU_S2CRn.TYPE field specifies that the initial context is Bypass mode, SMMU_S2CRn specifies memory attribute transformation for the transaction.

For translations that use a stage 1 translation context bank, SMMU_S2CRn specifies the first memory attribute transformation, and then either:

- If the SMMU_CBn_SCTLR.M bit is set to 0, that is, context bank MMU behavior is disabled, SMMU_CBn_SCTLR specifies the next attribute transformation applied.
- If the SMMU_CBn_SCTLR.M bit is set to 1, the translation table specifies the attributes.

When a stage 1 descriptor specifies the attributes, this overrides all attributes except those used for permission checking. Therefore, the only SMMU_S2CRn fields that affect such transactions are INSTCFG and PRIVCFG.

The resultant attributes are then combined with the stage 2 attributes, depending on the context bank format that the SMMU_CBARn.TYPE field specifies:

- For stage 1 context with stage 2 bypass format, the stage 1 context SMMU_CBARn specifies the stage 2 memory attributes.

- For stage 1 context with stage 2 context, the setting of the SMMU_CBn_SCTLR.M bit defines the mechanism for determining the stage 2 attributes. When the SMMU_CBn_SCTLR.M bit:
  - Is set to 0, the stage 2 context SMMU_CBn_SCTLR specifies the stage 2 attributes.
  - Is set to 1, the translation tables specify the stage 2 attributes.

If SMMU_S2CRn specifies that the initial context is translation context, and SMMU_CBARn specifies a stage 2 translation context bank, the stage 1 attributes are treated as the incoming attributes after the SMMU_S2CRn transformation has been applied. The stage 2 attributes are considered to be created in the same way as those for the stage 1 context with stage 2 context.

Stage 1 and stage 2 attributes are combined in accordance with ARM architectural requirements. The stage 2 attributes can only strengthen memory types, where the memory types are listed strongest to weakest:

- Strongly-ordered.
- Device.
- Non-cacheable to normal memory.
- Write-through to normal memory.
- Write-Back to normal memory.

In addition, for ARMv8 the Device memory type consists of the following Device memory types, listed strongest to weakest:

Device-nGnRnE

Device non-Gathering, non-Reordering, no Early Write Acknowledgement.

Equivalent to the Strongly-ordered memory type in ARMv7.

Device-nGnRE

Device non-Gathering, non-Reordering, Early Write Acknowledgement.

Equivalent to the Device memory type in ARMv7.

Device-nGRE

Device non-Gathering, Reordering, Early Write Acknowledgement.

ARMv8 adds this memory type to the translation table formats found in ARMv7. The use of barriers is required to order accesses to Device-nGRE memory.

Device-GRE

Device Gathering, Reordering, Early Write Acknowledgement.

ARMv8 adds this memory type to the translation table formats found in ARMv7. Device-GRE memory has the fewest constraints. It behaves similar to Normal memory, with the restriction that speculative accesses to Device-GRE memory is forbidden.

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

Similarly, the following shareabilities are listed strongest to weakest:

- Outer Shareable.
- Inner Shareable.
- Non-shareable.

In the ARM architecture, stage 2 attributes do not include a read or write allocation hints. In the SMMU architecture when context bank translation is disabled, the stage 2 SMMU_CBn_SCTLR RACFG and WACFG fields can be used with stage 1 translation. If the stage 2 SMMU_CBn_SCTLR.M bit is set to 1 then there are extra fields in the descriptor that specify RACFG and WACFG fields that are used. See Table 2-12 on page 2-84 for more information.

For memory type and shareability attributes, the following behaviors depend on the SMMU architecture version:

Combining Read and Write hint allocations

In SMMUv1, it is IMPLEMENTATION DEFINED how read and write allocate hints from stage 2 attributes combine with stage 1 attributes, as defined by the appropriate RACFG and WACFG fields.

In SMMUv2, the stage 2 attributes are used unless the stage 2 translation table format specifies that the value supplied from the previous stage is used, that is, if WACFG==`0b00` or RACFG==`0b00`.

Transforming bypassed transaction encodings

In SMMUv1, when the appropriate MTCFG field is set to 0, it is IMPLEMENTATION DEFINED whether certain bypassed client transactions have encodings that cannot be transformed. For example, an implementation might specify that incoming Strongly-ordered or Device transactions are not subject to any memory attribute transformation by RACFG or WACFG.

Note

- In such cases, the NSCFG field is guaranteed to be applied if Security state determination classes the transaction as Secure.
- If the appropriate MTCFG field is set to 1, the transformation is always applied, regardless of the encoding of the incoming transaction.

In SMMUv2, when the appropriate MTCFG field is set to 0, *Memory attribute, MemAttr on page 9-179* specifies the applicable encodings.

Instruction fetch configuration

In SMMUv1, it is IMPLEMENTATION DEFINED whether a SMMU_S2CRn.INSTCFG field transformation is applied to writes if that transformation would convert the transaction to an instruction write.

In SMMUv2, all writes are forced to `0b10`, indicating data.

2.4.1 Reserved memory type and shareability attributes

Many memory type and shareability attributes can be encoded as Reserved. In SMMUv2, these Reserved settings must be treated as shown in Table 2-4. In SMMUv1, the effect of using a Reserved value is UNPREDICTABLE.

Note

If software sets any of the attributes in Table 2-4 to a Reserved value, the Reserved value is returned on subsequent reads.

Table 2-4 Memory type and shareability attributes in SMMUv2

Attribute	Reserved values	SMMUv2 meaning
NSCFG	`0b01`	`0b11`, indicating Non-secure
WACFG	`0b01`	`0b00`, indicating default attributes
RACFG	`0b01`	`0b00`, indicating default attributes
MemAttr[3:0][a]	`0b0100`, `0b1000`, `0b1100`	`0b0000`, indicating Strongly-ordered or Device
BPSHCFG	`0b00`	`0b01`, indicating Outer Shareable
INSTCFG	`0b01`	`0b11`, indicating instruction
PRIVCFG[b]	`0b01`	`0b10`, indicating unprivileged
TRANSIENTCFG	`0b01`	`0b00`, indicating default attributes **Note** In systems that do not support transient marking of attributes, this attribute is RAZ/WI.

a. In SMMUv1, the values `0b0010` and `0b0011` are Reserved also.

b. This PRIVCFG value is reserved only when the value of SMMU_IDR2.DIPANS is 0.

2.4.2 Pseudocode for memory attribute operations

SMMU operations that are performed on memory attributes are:

- Combine operation, that outputs the stronger of two attribute types.
- Override operation, that applies a transformation serially from left to right, using the output of one transformation as the input to the next.

Note

- When any non-cacheable or Device attributes are mapped to cacheable memory by a MemAttr transformation, the pseudocode treats the memory as non-allocatable unless overridden by RACFG, WACFG, or TRANSIENTCFG.
- Hypervisor and Monitor contexts include a stage 2 attribute transformation stage even when the translation context is stage 1 context with stage 2 bypass. For the stage 2 attribute transformation to have no effect, software must set all transformations specified in SMMU_CBARn to use the default allocation attributes, with the following exceptions:
  - MemAttr must be set to `0b1111`, indicating Inner Write-Back and Outer Write-Back.
  - BPSHCFG must be set to `0b11`, indicating Non-shareable.

The following pseudocode generates final memory attributes.

```
// The following fields have reserved values, and in SMMUv1, if these reserved
// values are used the effects are UNPREDICTABLE.
// In SMMUv2, these reserved values should not used by the programmer but
// the hardware must interpret them as below:
// -    NSCFG, encoding 0b01.  Must be treated as 0b11 (Non-secure).
// -    WACFG, encoding 0b01.  Must be treated as 0b00 (Default Attributes)
// -    RACFG, encoding 0b01.  Must be treated as 0b00 (Default Attributes)
// -    TRANSIENTCFG, encoding 0b01.  Must be treated as 0b00 (Default Attributes).
//      Note: this field is RAZ/WI for bus systems that do not support transient marking.
// -    PRIVCFG, encoding 0b01.  Must be treated as 0b10 (Unprivileged).
// -    INSTCFG, encoding 0b01.  Must be treated as 0b11 (Instruction).
// -    BPSHCFG, encoding 0b00.  Must be treated as 0b01 (Outer shareable).
// -    MemAttr, encodings 0x4, 0x8 and 0xC.  Must be treated as 0x0 (Strongly Ordered)

attribute_transformations(attributes incoming)
{
    if (incoming is cache maintenance operation)
    {
        // A cache maintenance operation only has NS, shareability and whether the
        // operation must propagate to PointOfUnification or PointOfCoherency
        //
        // NOTE for AMBA systems, they also have:
        // - User or Privileged
        // - Instruction or Data
        // These interact with the permission model

        bool saved_PoU = incoming.IsPoU();

        incoming.inner = ("read allocate", "write allocate", "write back", "non transient");
        incoming.outer = ("read allocate", "write allocate", "write back", "non transient");
        // Note: NS, Inst, Priv and shareability are preserved

        attributes = data_attribute_transformations(incoming);

        // If the memory attributes end up being:
        // - Any device type
        // - inner non-cacheable, outer non-cacheable normal memory
        // Then architecturally these are outer shared and the cache maintenance
        // operation must be outer shared


        // NOTE: if the downstream bus system can represent any allocation or
```

```
            //       transient hints on a CMO then these are forced to zero.
            attributes.inner.read_allocate      = no_allocate;
            attributes.inner.write_allocate     = no_allocate;
            attributes.inner.transient_allocate = no_allocate;
            attributes.outer.read_allocate      = no_allocate;
            attributes.outer.write_allocate     = no_allocate;
            attributes.outer.transient_allocate = no_allocate;

            // Restore what where the operation should propagate to
            attributes.SetPoU(saved_PoU);

            if (IMP DEF option)
            {
                // It is implementation defined what Transient attribute
            // is marked on CMOs.  It may be defined by the bus standard
                // or it may be upto the implementation dependent on the SoC's
            // needs.
                attributes.SetInnerTransient(IMP DEF InnerTransient);
                attributes.SetOuterTransient(IMP DEF OuterTransient);
            }

            return attributes;
        }
        else if (incoming is barrier operation)
        {
            return barrier_attribute_transformation(incoming);
        }
        else
        {
            return data_attribute_transformations(incoming);
        }
    }

    attributes data_attribute_tranformations(attributes incoming)
    {
        if (CLIENTPD)
        {
            return Override(incoming, SMMU_sCR0);
        }

        switch (stream_match(incoming))
        {
        case unique:
            break;

        case multi_match:
            if (SMMU_sCR0.SMCFCFG)
            {
                // Stream Match Conflict Fault
                return FAULT;
            }
            return Override(incoming, SMMU_sCR0);

        case no_match:
            if (SMMU_sCR0.USFCFG)
            {
                // Unidentified Stream Fault
                return FAULT;
            }
            return Override(incoming, SMMU_sCR0);
        }

        switch (S2CR.TYPE)
        {
        case reserved:
            if (is_SMMUv1)
            {
                UNPREDICTABLE;
```

```
        }
        fallthrough;
    case fault:
        // Invalid Context Fault
        return FAULT;

    case bypass:
        return Override(incoming, S2CR);

    case translation:
        break;
    }

    if ( ! CheckBankValidityAndRecordFaults() )
    {
        return FAULT;
    }

    return Combine(
              Override(incoming, S2CR, Stage1Attributes()),
              Stage2Attributes()
              );

    // The above line reduces to the following expressions:
    // TYPE == st1_st2bypass:
    //   Combine( Override(incoming, s2cr, Stage1Attributes()), st1.cbar )
    // TYPE == st1_st2nested:
    //   Combine( Override(incoming, s2cr, Stage1Attributes()),
    //            (s2.sctlr.M ? s2.descriptor : s2.sctlr) )
    // TYPE == st2only:
    //   Combine( Override(incoming, s2cr),
    //            (s2.sctlr.M ? s2.descriptor : s2.sctlr) )
}

attributes barrier_attribute_tranformations(attributes incoming)
{
    // Barriers conceptually only have:
    // - shareability
    // - security state
    // - propagation point (Point of Unification / Point of Coherency)
    //
    // In AMBA systems they also have:
    // - Instruction not data
    // - Privileged not user
    //
    // NOTE everything except the propagation point can be transformed
    // NOTE barriers cannot be terminated by the SMMU unless unsupported
    //      downstream (in which case they are terminated if they don't fault)
    //
    // Barriers cannot generate:
    // - SIF faults
    // - Translation faults
    // - Address Size faults
    // - Permission faults
    // - Access faults
    //
    // Barriers can generate:
    // - Unidentified Stream Fault
    // - Stream Match Conflict Fault
    // - Invalid Context Fault
    // - Unimplemented Context Bank Fault
    //
    // Note: in SMMUv1 it is acceptable that even if a fault occurs, barriers
    // are always sent downstream.
    // However, the attribute transformation is UNPREDICTABLE.  This is
    // mostly harmless.

    if (CLIENTPD)
```

```
    {
        return BSU_Combine(incoming, SMMU_sCR0);
    }

    switch (stream_match(incoming))
    {
    case unique:
        break;
    case multi_match:
        if (SMMU_sCR0.SMCFCFG)
        {
            // Stream Match Conflict Fault
            return FAULT;
        }
        return BSU_Combine(incoming, SMMU_sCR0);
    case no_match:
        if (SMMU_sCR0.USFCFG)
        {
            // Unidentified Stream Fault
            return FAULT;
        }
        return BSU_Combine(incoming, SMMU_sCR0);
    }

    switch (S2CR.TYPE)
    {
    case fault:
        fallthrough;
    case reserved:
        // Invalid Context Fault
        return FAULT;

    case bypass:
        return BSU_Combine(incoming, S2CR);
    case translation:
        break;
    }

    if ( ! CheckBankValidityAndRecordFaults() )
    {
        return FAULT;
    }

    return BSU_Combine(
              BSU_Combine(incoming, S2CR, BarrierStage1Attributes()),
              BarrierStage2Attributes()
              );
}

Stage1Attributes()
{
    if (st1_st2bypass || st1_st2nested)
    {
        // In SMMUv2, for V7S the transient attribute is not transformed
        attribute_transform_t trans = s1.sctlr.M == 1 ? s1.descriptor :  s1.sctlr;

        // HYPC and MONC banks force privileged on the downstream bus if
        // supported by the bus.  It is IMP DEF if this forced attribute
        // is visible in the SMMU_sGFSYNR0.PNU or SMMU_CBn_FSYNR0.PNU.
        if (context bank is HYPC or MONC)
            trans.PRIVCFG = "Privileged";

        return trans;
    }
    else
    {
        return <no-attributes>;
    }
```

```
}

BarrierStage1Attributes()
{
    if (st1_st2bypass || st1_st2nested)
    {
        return s1.sctlr;
    }
    else
    {
        return <no-attributes>;
    }
}

Stage2Attributes()
{
    if (st1_st2bypass)
    {
        return s1.cbar;
    }
    return s2.sctlr.M == 1 ? s2.descriptor : s2.sctlr;
}

BarrierStage2Attributes()
{
    if (st1_st2bypass)
    {
        return s1.cbar;
    }
    return s2.sctlr;
}

// The complete attributes form a total order where one value is stronger than
// another:
//
// Shareability  :  osh > ish > nsh
// Memory type   :  nGnRnE > nGnRE > nGRE > GRE  > nc_nb > nc  > wt > wb
// Security state:  Non-secure > Secure
//
// Secure can ONLY be downgraded, hence non-secure > secure
//
attributes StrongerOf( attributes s1_attributes, attributes s2_attributes );

attributes Combine( attributes s1_attributes, attributes s2_attributes )
{
     attributes attributes;

     // For each of the attribute fields then choose the stronger of the
     // two.
     foreach field F in attributes
     {
          if (F in {read_allocate, write_allocate, transient_allocate})
          {
             attributes.F = Combine_allocate( s1_attributes.F, s2_attributes.F );
          }
          else
          {
             attributes.F = StrongerOf( s1_attributes.F, s2_attributes.F );
          }
     }

     //
     // Normalize attributes
     //
     // We can end up with strange attributes that do not obey the
     // architectural properties required.  This stage normalizes these
     // cases
     //
```

```
    if (is_any_device(is_SMMUv1, attributes)
        || is_inner_and_outer_non_cacheable_normal(attributes))
    {
        attributes.shareability             = outer_shared;
        attributes.inner.read_allocate      = no_allocate;
        attributes.inner.write_allocate     = no_allocate;
        attributes.inner.transient_allocate = no_allocate;
        attributes.outer.read_allocate      = no_allocate;
        attributes.outer.write_allocate     = no_allocate;
        attributes.outer.transient_allocate = no_allocate;
    }
    else
    {
        if (attributes.inner is not cacheable)
        {
            attributes.inner.read_allocate      = no_allocate;
            attributes.inner.write_allocate     = no_allocate;
            attributes.inner.transient_allocate = no_allocate;
        }

        if (attributes.outer is not cacheable)
        {
            attributes.outer.read_allocate      = no_allocate;
            attributes.outer.write_allocate     = no_allocate;
            attributes.outer.transient_allocate = no_allocate;
        }
    }

    return attributes;
}

Combine_allocate( s1_attributes, s2_attributes )
{
    // There is no intuitive 'stronger-than' relation for allocation and so we
    // choose to always obey the s2 attributes if they are anything other than
    // 'no-transform'.  The rationale is that the hypervisor should have more
    // SoC-specific knowledge than the guest OS and should know if it is
    // appropriate to allocate into the downstream cache or not (weak argument).
    //
    // SMMUv1 did not specify this behaviour (making it IMPLEMENTATION DEFINED).
    // In SMMUv2 the behavior is specified.
    //
    if (s2_attributes != 0b00 /*no-transform*/)
    {
        return s2_attributes; // Ignore s1_attributes
    }
    return s1_attributes;
}

MemoryAttributes map_MemAttr_to_attributes[16] = {
           // Outer      Inner
           // -----      -----
/* 00 */    { so,        so    },
/* 01 */    { dv,        dv    },
/* 02 */    { nGRE,      nGRE  },  // new to V8, UNPRED in v7
/* 03 */    { GRE,       GRE   },  // new to V8, UNPRED in v7

/* 04 */    { BAD,       BAD   },
/* 05 */    { nc,        nc    },
/* 06 */    { nc,        naWT  },
/* 07 */    { nc,        naWB  },

/* 08 */    { BAD  ,     BAD   },
/* 09 */    { naWT,      nc    },
/* 10 */    { naWT,      naWT  },
/* 11 */    { naWT,      naWB  },

/* 12 */    { BAD  ,     BAD   },
```

```
/* 13 */    { naWB,      nc    },
/* 14 */    { naWB,      naWT  },
/* 15 */    { naWB,      naWB  }
};

transaction transform_by_register(register reg, transaction incoming)
{
    // NOTE that not all fields are present in all registers, but we present
    // them as all present.  In these cases then the corresponding field will
    // be treated as 'use value supplied from previous attribute transformation step'.
    //
    // NOTE not all bus systems signal both inner and outer attributes.  It is
    // IMP DEF which set will be used (but will be consistent throughout the
    // device).
    //

    if (reg.MTCFG)
    {
        // v8 adds new device types so is_any_device() needs to know the
        // architecture version to determine if it is any type of device

        attributes = map_MemAttr_to_attributes[reg.MemAttr];

        if (attributes == BAD
            || (is_SMMUv1 && (attributes is nGRE or GRE)))
        {
            if (is_SMMUv1)
            {
                attributes = UNPREDICTABLE; // maybe FAULT
            }
            else
            {
                // Treat as strongly ordered
                attributes = { so, so };
            }
        }

        //
        // NOTE that if the original transaction was non-cacheable then
        // the MemAttr field encodes no-allocate.
        //
        // If the incoming transaction is cacheable then the allocation attributes
        // will come from the incoming transaction
        //
        if (incoming.attributes().inner is cacheable
            && attributes.inner is cacheable)
        {
            attributes.inner.read_allocate      = incoming.attributes().inner.read_allocate;
            attributes.inner.write_allocate     = incoming.attributes().inner.write_allocate;
            attributes.inner.transient_allocate = incoming.attributes().inner.transient_allocate;
        }

        if (incoming.attributes().outer is cacheable
            && attributes.outer is cacheable)
        {
            attributes.outer.read_allocate      = incoming.attributes().outer.read_allocate;
            attributes.outer.write_allocate     = incoming.attributes().outer.write_allocate;
            attributes.outer.transient_allocate = incoming.attributes().outer.transient_allocate;
        }
    }
    else
    {
        // Not MTCFG
        attributes = incoming.attributes();
    }

    if (is_any_device(is_SMMUv1, attributes) || is_iNC_oNC(is_SMMUv1, attributes))
    {
```

```
    // Device
    shareability         = outer_shared;

    inner_read_allocate  = false;
    inner_write_allocate = false;
    inner_transient      = false;

    outer_read_allocate  = false;
    outer_write_allocate = false;
    outer_transient      = false;
}
else
{
    // Normal memory

    shareability         = reg.SHCFG( incoming.shareability() );

    if (attributes.inner_is_cacheable())
    {
        inner_read_allocate = reg.RACFG(attributes.inner.read_allocate );
        inner_write_allocate = reg.WACFG(attributes.inner.write_allocate );
        inner_transient = reg.TRANSIENT(attributes.inner.transient_allocate );
    }
    else
    {
        inner_read_allocate  = false;
              inner_write_allocate = false;
              inner_transient      = false;
    }



    if (attributes.outer_is_cacheable())
    {
        outer_read_allocate = reg.RACFG(attributes.outer.read_allocate );
        outer_write_allocate = reg.WACFG(attributes.outer.write_allocate );
        outer_transient = reg.TRANSIENT(attributes.outer.transient_allocate );
    }
    else
    {
        outer_read_allocate  = false;
              outer_write_allocate = false;
              outer_transient      = false;
    }
}

ns = ssd_is_non_secure ? NON_SECURE : reg.NSCFG(incoming.ns() );

if (is_read)
{
    inst = reg.INSTCFG(incoming.inst() );
}
else
{
    inst = "data";  // All writes are represented as Data (not instruction) writes
                    // even if the upstream system marks it as Instruction
}

if (is_SMMUv1 && ! is_read &&  reg.INSTCFG == instcfg_instruction)
{
    inst = IMP_DEF;
}

priv = reg.PRIVCFG(incoming.priv() );

// Set up the outgoing transaction

Transaction outgoing = incoming;
```

```
    outgoing.attributes
        = (inst, priv, ns
           shareability,
           attributes,
           inner_read_allocate, inner_write_allocate, inner_transient,
           outer_read_allocate, outer_write_allocate, outer_transient );

    return outgoing;
}

transaction Override(transaction incoming, register[] regs)
{
    transaction outgoing = incoming;

    foreach(register reg in regs)
    {
        outgoing = transform_by_register(outgoing, reg);
    }

    return outgoing;
}

transaction BSU_Combine(transaction incoming, register[] regs)
{
    transaction outgoing = incoming;

    foreach(register reg in regs)
    {
        outgoing.shareability = StrongerOf(reg.BSU, outgoing.shareability);
    }

    return outgoing;
}
```

2.5 TLB operation

The SMMU architecture supports the use of TLBs to accelerate the translation process, and provides TLB maintenance operations to manage the TLBs. See *TLB maintenance operations on page 5-137* for more information. If a TLB does not contain an appropriate translation, then the SMMU uses translation tables to translate the address.

The result of a successful TLB lookup contains an output address, memory permission, and memory attribute information. TLB entries hold information relating to the context of the entry. This information, often described as TLB *tags*, is required for TLB invalidation.

Note

- The exact behavior of any TLB functionality is IMPLEMENTATION DEFINED.
- TLB entries can cache data for any level and stage or combination of levels and stages of the translation process. The SMMU architecture does not limit how a TLB is implemented, provided it obeys the TLB Invalidate operations.

TLB maintenance operations can be initiated either by:

- Broadcast TLB operations, as described in *Broadcast TLB maintenance operations on page 5-136*.
- Accessing TLB maintenance registers, as described in *TLB maintenance operations on page 5-137*.

A TLB entry, or combination of TLB entries, provides the required translation address and the memory attribute information for that address. The tags used in the lookup determine both:

- The set of TLB entries that are considered when the SMMU attempts to locate a translation.
- The entry, or entries, that the SMMU selects when processing a TLB Invalidate operation.

Software must treat all TLB entries as being tagged with the input address. For other attributes, TLB entries are categorized so that the tagging applied to an entry depends on the translation regime. For a particular translation regime, a TLB entry must specify appropriate information, and certain conditions must be satisfied. For example, a Non-secure stage 1 TLB entry must specify an ASID. Table 2-5 shows the types of TLB entry that the SMMU architecture supports, and the corresponding attribute tagging requirements.

Table 2-5 SMMU TLB entry attribute requirements

Translation context	Global?	Attribute required?		Notes
		ASID[a]	VMID	
HYPC	-	-	-	See *Hypervisor contexts (HYPC) on page 2-86*
MONC	-	-	-	See *Monitor contexts (MONC) on page 2-89*
E2HC	N	Y	-	See *E2H contexts (E2HC) on page 2-90*
	Y	N	-	
Non-secure stage 1 with stage 2 bypass	N	Y	Y[b]	-
	Y	N	Y[b]	-
Non-secure stage 1 followed by stage 2	N	Y	Y[b]	-
	Y	N	Y[b]	-
Secure stage 1	N	Y	N	-
	Y	N	N	-
Non-secure stage 2	-	N	Y[c]	The TLB entries are stage 2 only TLB entries

a. Also applies to ASIDPNE tagging when use of ASIDPNE is supported. See *Address Space Identifiers, ASIDs on page 1-38* for more information.
b. The VMID comes from either the stage 1 or stage 2 context bank SMMU_CBARn.VMID. The stage 1 and the stage 2 context bank must have the same VMID, or the result is UNPREDICTABLE. If SMMU_CR0.VMID16EN is set to 1, then see *Virtual machine identifiers, VMIDs on page 1-39*.
c. The VMID comes from the stage 2 context bank SMMU_CBARn.VMID. If 16-big VMIDs are in use, then see *Virtual machine identifiers, VMIDs on page 1-39*.

If the SMMU implementation does not support transient marking, there is no requirement for TLBs to include any transient information.

Note

When an implementation does not support transient marking, the SMMU_S2CRn.TRANSIENTCFG field is RO, with value `0b00`.

TLB entries are not required to specify a context bank identifier. This means that, in general, all context banks that have the same translation context can share TLB entries, and a TLB invalidation operation affects all context banks that have the same attributes.

Note

Typically, all context banks that have the same translation context share TLB entries. However, such sharing of TLB entries is not an SMMU architectural requirement.

If multiple context banks have the same attributes but describe different translations, the results of a TLB lookup are UNPREDICTABLE. For example, where two context banks use the same ASID and VMID, each context bank might use TLB entries that are intended to be used by the other context bank.

An SMMUv2 implementation might provide IMPLEMENTATION DEFINED mechanisms to lock and unlock particular TLB entries. The SMMU cannot invalidate locked TLB entries.

The SMMU architecture permits any TLB invalidation operation to invalidate more TLB entries than the operation specifies, provided the entries are not locked. The extent to which an operation over-invalidates is IMPLEMENTATION DEFINED.

Software can use SMMU_sTLBGSTATUS to verify completion of all the TLB invalidate operations initiated before the most recent write to SMMU_sTLBGSYNC. See *TLB maintenance operation processing on page 5-139* for more information.

It is IMPLEMENTATION DEFINED whether an SMMU supports broadcast TLB maintenance operations. See *Broadcast TLB maintenance operations on page 5-136* for more information.

2.5.1 Prefetch behavior for TLB entries

If transactions are using a context bank, the SMMU is permitted to prefetch TLB entries for that context, even if the prefetched entries are not related to any transaction using that context that has been received by the SMMU. This means that when a context bank is quiescent, there is no guarantee that the SMMU stops prefetching TLB entries that might be used by that context bank. There is therefore a risk of prefetching invalid TLB entries. To prevent this, the SMMU architecture places constraints on prefetch behavior. See *Updating the state of a context bank on page 5-134* for more information.

2.5.2 Reset behavior of the TLB

In SMMUv2:

- All TLBs are disabled from reset. All MMUs are disabled from reset, and the contents of the TLBs have no effect on address translation.

- An implementation can require the use of a specific TLB invalidation routine, to invalidate the TLB arrays before they are enabled after a reset. The exact form of this routine is IMPLEMENTATION DEFINED, but if an invalidation routine is required, it must be documented clearly as part of the documentation of the device.

  ARM recommends that if an invalidation routine is required for this purpose, the routine is based on the SMMUv2 TLB maintenance operations described in *TLB maintenance operations on page 5-137*.

- When TLBs that have not been invalidated by some mechanism since reset are enabled, the state of those TLBs in UNPREDICTABLE.

2.6 Translation context

A translation context provides information and resources required by the SMMU to process a transaction.

The SMMU can process multiple transaction streams from different threads of execution, and supports multiple live translation contexts.

A translation context bank includes state for configuring the translation process and capturing fault status, and operations for maintaining cached translations. A translation context bank specifies largely the same state used by the ARM processor architecture translation process, principally:

- The translation table base addresses.
- Memory attributes to use during the translation table walk.
- Translation table attribute remapping.

The format of a translation context bank depends on whether it is used for stage 1 or stage 2 translation.

Context bank determination determines the translation context bank that is to be used during the processing of a transaction. Up to two translation context banks can be specified:

- In implementations that do not support stage 1 followed by stage 2 translations, only one translation context bank can be specified.
- In implementations that support stage 1 followed by stage 2 translations:
  - One translation context bank is specified for single-stage translation.
  - Two translation context banks are specified for two-stage translation.

2.6.1 The translation context bank table

Translation context banks are arranged as a table in the SMMU configuration address map. Each entry in the table occupies a PAGESIZE address space, where PAGESIZE is 4KB or 64KB. See *PAGESIZE and NUMPAGENDXB on page 8-164*.

The SMMU architecture provides space for up to 128 translation context banks. The PAGESIZE alignment means a hypervisor can give a Guest OS direct access to a specific translation context bank by using the stage 2 translation functionality provided by an MMU on the processor. Similarly, a hypervisor can use address translation to give the illusion of a contiguous translation context bank table to a Guest OS.

A single translation context bank table contains all the translation context banks, regardless of whether they are stage 1 or stage 2 format.

The number of translation context banks an SMMU implements is IMPLEMENTATION DEFINED, and is specified by SMMU_IDR1.NUMCB. An attempt to access an unimplemented translation context bank results in one of the following IMPLEMENTATION DEFINED behaviors:

- The access is treated as RAZ/WI.
- The access generates a configuration access fault, as Chapter 3 *The Fault Model* describes.

Some translation context banks might only support stage 2 translations.

SMMU_IDR1.NUMS2CB specifies the IMPLEMENTATION DEFINED number of translation context banks that only support stage 2 translation. The remaining translation context banks support either stage 1 or stage 2 translation. Translation context banks that only support stage 2 translation are grouped together, starting at location 0 in the translation context bank table. The remaining translation context banks are similarly grouped and start immediately after the translation context banks that only support stage 2 translations.

For translation context banks that can be used for either stage 1 or stage 2 translations, the translation format is specified by the SMMU_CBARn.TYPE field associated with the translation context bank.

Translation context bank table in an implementation that supports two Security states

In an implementation that supports two Security states, SMMU_SCR1.NSNUMCBO can be used to allocate a number of translation context banks for Secure use. This field allocates translation context banks from the top of the translation context bank table. Because only stage 1 translation is supported for Secure transactions, only translation context banks that can support stage 1 translation can be allocated for Secure use.

An SMMU_S2CRn register that is allocated for use by Secure software must specify a translation context bank of appropriate security, as *The Stream mapping table in an implementation that supports two Security states on page 2-58* describes.

For configuration accesses, Secure software can access all translation context banks, and Non-secure software can only access translation context banks that are not allocated for use by Secure software. This means Secure software can inspect Non-secure translation context banks.

The value written to SMMU_SCR1.NSNUMCBO affects the value read from SMMU_IDR1.NUMCB by Non-secure register accesses.

Note

In SMMUv2, when a context bank changes Security state, SMMU_CBA2Rn.VA64 becomes UNKNOWN. A consequence is that SMMUv1 Secure software that changes SMMU_SCR1.NSNUMCBO must be updated, or it might result in a security violation. SMMUv2 compliant Secure software has to always initialize this bit. See *Resource allocation on page 7-158* for more information.

Figure 2-6 gives an overview of the translation context bank table, SMMU_IDR1 fields and the effect of the override registers.

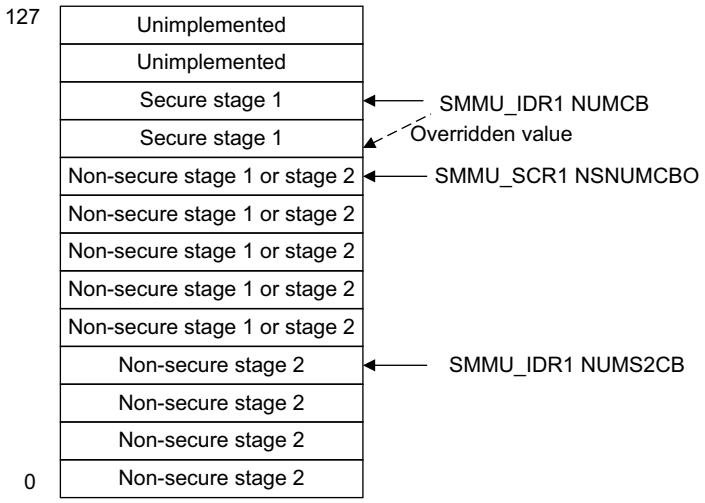


Figure 2-6 Translation context bank table

Selection of translation scheme and TTBR use

How a transformation is processed using a particular context bank is determined by:

- The Security state of the bank.
- The SMMU_CBARn.TYPE field.
- The SMMU_CBA2Rn.VA64 bit, if the processing requires an address translation.
- The SMMU_CBARn.HYPC bit.
- For Secure banks, the SMMU_CBA2Rn.MONC bit.

Table 2-6 shows the effect on TTBR determination of the SMMU_CBA2Rn.VA64 and SMMU_CBARn.HYPC bits for Non-secure translation context banks.

Table 2-6 TTBR controls for Non-secure translation context banks

VA64 bit	HYPC bit	Translation stage	TTBR	Description
0	0	Stage 1	TTBR0 and TTBR1	A Non-secure 32-bit OS translation context
0	1	Stage 1	TTBR0[a]	A Non-secure 32-bit Hypervisor translation context
0	-	Stage 2	TTBR0	A Non-secure stage 2 translation context
1	0	Stage 1	TTBR0 and TTBR1	A Non-secure 64-bit OS translation context
1	1	Stage 1	TTBR0	A Non-secure 64-bit Hypervisor translation context
1	-	Stage 2	TTBR0	A Non-secure stage 2 translation context

a. Although the value of the SMMU_CBn_TCR.EAE bit is ignored, ARM recommends that it is set to 1.

Table 2-7 shows the effect on TTBR determination of the SMMU_CBA2Rn.{VA64, MONC} and SMMU_CBARn.HYPC bits for Secure translation context banks.

Table 2-7 TTBR controls for Secure translation context banks

VA64 bit	HYPC bit	MONC bit	Translation stage	TTBR	Description
0	0	-	Stage 1	TTBR0 and TTBR1	A Secure 32-bit OS translation context
	1	-	Stage 1	TTBR0	A Secure[a] 32-bit Hypervisor translation context
1	0	0	Stage 1	TTBR0 and TTBR1	A Secure 64-bit OS translation context
		1	Stage 1	TTBR0	A Secure 64-bit Monitor translation context
	1	0	Stage 1	TTBR0	A Secure[a] 64-bit Hypervisor translation context
		1	Stage 1	-	UNPREDICTABLE

a. Although a hypervisor exists only in Non-secure state in the ARM architecture, Secure Hypervisor contexts are specific to the SMMU architecture. See *Hypervisor contexts (HYPC) on page 2-86*.

Translation context bank for stage 1 translation

A translation context bank for stage 1 translation contains the following:

- Context control registers.
- Translation table base and control registers.
- Fault address and status registers.
- OPTIONAL address translation registers that permit software to initiate address translation operations.
- Physical address and status registers.
- Registers that initiate TLB maintenance operations.

See Chapter 16 *Stage 1 Translation Context Bank Format* for more details.

Translation context bank for stage 2 translation

The format of a translation context bank configured for stage 2 translation is similar to the stage 1 format. See *Stage 1 and stage 2 context bank format differences on page 17-344* for information about the differences, and see Chapter 17 *Stage 2 Translation Context Bank Format* for more information about the stage 2 format.

Reset state

On reset, no translation context bank table entry initialization is required. Typically, the SMMU implementation initializes the required values before use.

2.6.2 The Context Bank Attribute Register, SMMU_CBAR*n*

Each translation context bank has an associated SMMU_CBARn register. A translation context bank of index *n* is associated with an SMMU_CBARn. This register provides additional attributes for the translation context bank. This means an SMMU implementation must implement the same number of SMMU_CBARn registers as the number of implemented translation context banks. Each SMMU_CBARn is part of the corresponding translation context bank, but has a different access mechanism.

If SMMU_CBARn configures the associated translation context bank for stage 1 translation, it can specify one of the following additional contexts:

- In an implementation that supports stage 1 followed by stage 2 translation, a second translation context bank.
- Bypass mode, performing no stage 2 translation or address protection, but applying stronger memory attributes than those applied by the stage 1 translation, in a similar way to the possible strengthening of memory attributes by a stage 2 translation context bank. See *Memory type and shareability attribute determination on page 2-59* for more information.

In SMMUv1, if a translation context bank is configured to raise an interrupt in the event of a context fault, SMMU_CBARn specifies the interrupt to be asserted if the context fault occurs. In SMMUv2, each context bank has a dedicated interrupt pin. See *Context interrupts on page 3-100* for more information.

The SMMU_CBARn registers are organized as a table in the SMMU configuration address space. A configuration access to an unimplemented SMMU_CBARn register results in one of the following IMPLEMENTATION DEFINED behaviors:

- The access is treated as RAZ/WI.
- The access generates a configuration access fault, as Chapter 3 *The Fault Model* describes.

SMMU_CBARn.VMID specifies the Virtual Machine Identifier for the corresponding translation context bank. When associated with a Non-secure, non-hypervisor stage 1 translation context bank, the VMID field is used as follows:

- The VMID is associated with a transaction being translated using this translation context bank. The VMID is used for TLB tagging and matching purposes.
- The VMID is used for TLB maintenance operations issued in the corresponding translation context bank. The VMID is used for TLB matching purposes.
- If the context is stage 1 followed by stage 2, then the stage 1 SMMU_CBARn.VMID and the stage 2 SMMU_CBARn.VMID must be identical, or the result is UNPREDICTABLE.

For both stage 1 and stage 2 translation context banks, if the implementation supports 16-bit VMIDs, that is, if SMMU_IDR2.VMID16S is 1:

- When SMMU_CR0.VMID16EN is 0, then 8-bit VMIDs are in use, and SMMU_CBARn.VMID specifies the VMID, and SMMU_CBA2Rn.VMID16 is RES0.
- When SMMU_CR0.VMID16EN is 1, then 16-bit VMIDs are in use, and SMMU_CBA2Rn.VMID16 holds the 16-bit VMID and SMMU_CBARn.VMID is RES0.

For simplicity, all statements in the architecture that indicate the VMID comes from SMMU_CBARn.VMID should also be read to indicate that the VMID comes from SMMU_CBA2Rn.VMID16 when support for 16-bit VMIDs is enabled.

See *SMMU_CBARn, Context Bank Attribute Registers on page 10-241* for more information.

Note

The use of VMID is modified for:

- Secure context banks, as this section describes.
- Hypervisor contexts. See *TLB maintenance for HYPC banks on page 2-87*.

In an implementation that supports two Security states, using SMMU_SCR1.NSNUMCBO to allocate a translation context bank for Non-secure use also allocates the corresponding SMMU_CBARn register. This also determines the number of SMMU_CBARn registers visible to Non-secure software.

Secure software must only use *Stage 1 context with stage 2 bypass, TYPE==0b01 on page 10-242*. Using any other type generates an invalid context fault.

A VMID can be associated with Secure transactions in both translation and bypass contexts. This provides support in legacy systems where VMID is propagated to the endpoint in the system, as an attribute to the transaction, at which point more checks are made. When associated with a Secure transaction:

- A VMID has no involvement in translation matching.
- It is IMPLEMENTATION DEFINED whether the VMID is visible in translation table walks and client transactions.

Reset state

On reset, the SMMU_CBARn registers are not initialized. They must be initialized before use.

2.6.3 The translation tables

Translation tables specifying the output address and memory attributes for a context are stored in memory. The SMMU_CBn_TTBRm register, or registers, specify the translation table base address for a corresponding translation context, where *m* is 1 or 0. See:

- *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339*.
- *SMMU_CBn_TTBR0, Translation Table Base Register on page 17-360*.

2.6.4 The algorithm for finding the translation table descriptors

For translations using 64-bit translation table descriptors, this section gives the algorithms for finding the translation table descriptor that corresponds to a given IA, for each required level of lookup. The algorithm details depend on the translation granule size for the stage of address translation, see:

- For AArch32 Long-descriptor translation scheme:
  - *Finding the translation table descriptor when using the VMSAv8-32 Long-descriptor format on page 2-80*.
- For AArch64 translation regime schemes:
  - *Finding the translation table descriptor, VMSAv8-64 using 4KB translation granule on page 2-81*.
  - *Finding the translation table descriptor, VMSAv8-64 using 16KB translation granule on page 2-82*.
  - *Finding the translation table descriptor, VMSAv8-64 using 64KB translation granule on page 2-83*.

Each subsection uses the following terms:

BaseAddr The base address for the level of lookup, as defined by:

- For the initial lookup level, the appropriate TTBR.
- Otherwise, the translation table address returned by the previous level of lookup.

PAMax The supported PA width, in bits.

IA The supplied IA for this stage of translation.

T*n*SZ The translation table size for this stage of translation. See SMMU_CBn_TCR.T*n*SZ.

SL0 SMMU_CBn_TCR.SL0. Applies to the Non-secure EL1&0 stage 2 translation only.

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

Finding the translation table descriptor when using the VMSAv8-32 Long-descriptor format

The VMSAv8-32 Long Descriptor format uses a 4KB translation granule. Table 2-8 shows the translation table descriptor address, for each level of lookup, when using the VMSAv8-32 Long-descriptor format. See the start of *The algorithm for finding the translation table descriptors on page 2-79* for more information about terms used in the table.

Table 2-8 Translation table entry addresses, VMSAv8-32 using Long-descriptor format

Lookup level	Entry address and conditions		General conditions
	Stage 1 translation	Stage 2 translation	
One	BaseAddr[39:*x*]:IA[*y*:30]:0b000 if[a] 0 ≤ T*n*SZ ≤ 1 then *x* = (5 - T*n*SZ)	BaseAddr[39:*x*]:IA[*y*:30]:0b000 if[a] -8 ≤ T0SZ ≤ 1 then *x* = (5 - T0SZ)	*y* = (*x* + 26)
Two	BaseAddr[39:*x*]:IA[*y*:21]:0b000 if 2 ≤ T*n*SZ ≤ 7then *x* = (14 - T*n*SZ) else[b] *x* =12	BaseAddr[39:*x*]:IA[*y*:21]:0b000 if SL0 == 0 then if[a] -2 ≤ T0SZ ≤ 7 then *x* = (14 - T0SZ) elsif[b] SL0 == 1 then *x* = 12	*y* = (*x* + 17)
Three	BaseAddr[39:12]:IA[20:12]:0b000	BaseAddr[39:12]:IA[20:12]:0b000	-

a. This line indicates the range of permitted values for T*n*SZ, for a lookup that starts at this level.
b. This is the case where this level of lookup is not the initial level of lookup.

Finding the translation table descriptor, VMSAv8-64 using 4KB translation granule

Table 2-9 shows the translation table descriptor address, for each level of lookup, when VMSAv8-64 is using the 4KB translation granule. See the start of *The algorithm for finding the translation table descriptors on page 2-79* for more information about terms used in the table.

Table 2-9 Translation table entry addresses when using the 4KB translation granule

Lookup level	Entry address and conditions		General conditions
	Stage 1 translation	Stage 2 translation	
Zero	BaseAddr[PAMax-1:*x*]:IA[*y*:39]:0b000 if[a] 16 ≤ T*n*SZ ≤ 24 then *x* = (28 - T*n*SZ)	BaseAddr[PAMax-1:*x*]:IA[*y*:39]:0b000 if SL0[b] == 2 then if[a] 16 ≤ T0SZ ≤ 24 then *x* = (28 - T0SZ)	*y* = (*x* + 35)
One	BaseAddr[PAMax-1:*x*]:IA[*y*:30]:0b000 if[a] 25 ≤ T*n*SZ ≤ 33 then *x* = (37 - T*n*SZ) else[c] *x* =12	BaseAddr[PAMax-1:*x*]:IA[*y*:30]:0b000 if SL0[b] == 1 then if[a] 21 ≤ T0SZ ≤ 33 then *x* = (37 - T0SZ) elsif SL0[b, c]== 2 then *x* = 12	*y* = (*x* + 26)
Two	BaseAddr[PAMax-1:*x*]:IA[*y*:21]:0b000 if[a] 34 ≤ T*n*SZ ≤ 39 then *x* = (46 - T*n*SZ) else[c] *x* =12	BaseAddr[PAMax-1:*x*]:IA[*y*:21]:0b000 if SL0[b] == 0 then if[a] 30 ≤ T0SZ ≤ 39 then *x* = (46 - T0SZ) elsif SL0[b, c] > 0 then *x* = 12	*y* = (*x* + 17)
Three	BaseAddr[PAMax-1:12]:IA[20:12]:0b000	BaseAddr[PAMax-1:12]:IA[20:12]:0b000	-

a. This line indicates the range of permitted values for T*n*SZ, for a lookup that starts at this level.
b. SL0 == 0 if the initial lookup is level 2, SL0 == 1 if the initial lookup is level 1, and SL0 ==2 if the initial lookup level is level 0.
c. This is the case where this level of lookup is not the initial level of lookup.

Finding the translation table descriptor, VMSAv8-64 using 16KB translation granule

Table 2-10 shows the translation table descriptor address, for each level of lookup, when VMSAv8-64 is using the 16KB translation granule. See the start of *The algorithm for finding the translation table descriptors on page 2-79* for more information about terms used in the table.

Table 2-10 Translation table entry addresses when using the 16KB translation granule

Lookup level	Entry address and conditions		General conditions
	Stage 1 translation	Stage 2 translation	
Zero	BaseAddr[PAMax-1:4]:IA[47]:0b000 [a] 16 ≤ T*n*SZ	-	Only applies to stage 1
One	BaseAddr[PAMax-1:*x*]:IA[*y*:36]:0b000 if [a] 17 ≤ T*n*SZ ≤ 27 then *x* = (31 - T*n*SZ) else [b] *x* =14	BaseAddr[PAMax-1:*x*]:IA[*y*:36]:0b000 if SL0 [c] == 2 then if [a] 16 ≤ T0SZ ≤ 27 then *x* = (31 - T0SZ)	*y* = (*x* + 32)
Two	BaseAddr[PAMax-1:*x*]:IA[*y*:25]:0b000 if [a] 28 ≤ T*n*SZ ≤ 38 then *x* = (42 - T*n*SZ) else [b] *x* =14	BaseAddr[PAMax-1:*x*]:IA[*y*:25]:0b000 if SL0 [c] == 1 then if [a] 24 ≤ T0SZ ≤ 38 then *x* = (42 - T0SZ) elsif SL0 [b, c] == 2 then *x* = 14	*y* = (*x* + 21)
Three	BaseAddr[PAMax-1:14]:IA[24:14]:0b000	BaseAddr[PAMax-1:*x*]:IA[*y*:14]:0b000 if SL0 [c] == 0 then if [a] 35 ≤ T0SZ ≤ 39 then *x* = (53 - T0SZ) elsif SL0 [b, c] >0 then *x* = 14	*y* = (*x* + 10)

a. This line indicates the range of permitted values for T*n*SZ, for a lookup that starts at this level.
b. This is the case where this level of lookup is not the initial level of lookup.
c. SL0 == 0 if the initial lookup is level 3, SL0 == 1 if the initial lookup is level 2, and SL0 ==2 if the initial lookup level is level 1.

Finding the translation table descriptor, VMSAv8-64 using 64KB translation granule

Table 2-11 shows the translation table descriptor address, for each level of lookup, when VMSAv8-64 is using the 64KB translation granule. See the start of *The algorithm for finding the translation table descriptors on page 2-79* for more information about terms used in the table.

Table 2-11 Translation table entry addresses when using the 64KB translation granule

Lookup level	Entry address and conditions		General conditions
	Stage 1 translation	**Stage 2 translation**	
One	BaseAddr[PAMax-1:*x*]:IA[*y*:42]:`0b000` if[a] 16 ≤ T*n*SZ ≤ 21 then *x* = (25 - T*n*SZ)	BaseAddr[PAMax-1:*x*]:IA[*y*:42]:`0b000` if SL0[b] == 2 then if[a] 16 ≤ T0SZ ≤ 21 then *x* = (25 - T0SZ)	*y* = (*x* + 38)
Two	BaseAddr[PAMax-1:*x*]:IA[*y*:29]:`0b000` if[a] 22 ≤ T*n*SZ ≤ 34 then *x* = (38 - T*n*SZ) else[c] *x* =16	BaseAddr[PAMax-1:*x*]:IA[*y*:29]:`0b000` if SL0[b] == 1 then if[a] 18 ≤ T0SZ ≤ 34 then *x* = (38 - T0SZ) elsif SL0[b, c] == 2 then *x* = 16	*y* = (*x* + 25)
Three	BaseAddr[PAMax-1:*x*]:IA[*y*:16]:`0b000` if[a] 35 ≤ T*n*SZ ≤ 39 then *x* = (51 - T*n*SZ) else[c] *x* =16	BaseAddr[PAMax-1:*x*]:IA[*y*:16]:`0b000` if SL0[b] == 0 then if[a] 31 ≤ T0SZ ≤ 39 then *x* = (51 - T0SZ) elsif SL0[b, c] > 0 then *x* = 16	*y* = (*x* + 12)

a. This line indicates the range of permitted values for T*n*SZ, for a lookup that starts at this level.

b. SL0 == 0 if the initial lookup is level 3, SL0 == 1 if the initial lookup is level 2, and SL0 ==2 if the initial lookup level is at level 1.

c. This is the case where this level of lookup is not the initial level of lookup.

2.7 Translation and protection checks

If SMMU_S2CRn specifies a translation context, a translation table walk retrieves translation and memory attribute information for the transaction. The transaction might require two stages of translation, as configured in the Stream mapping table and SMMU_CBARn registers.

2.7.1 Translation table format

The SMMU translation table formats and translation process are generally the same as those in described in the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*, for the AArch32 and AArch64 translation regimes. The AArch32 formats and processes are compatible with those for ARMv7, as described in the *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition*. *Stage 2 translation table format* describes an SMMU extension to those formats.

Stage 2 translation table format

The ARM processor architecture reserves bits [63:60] of the stage 2 translation table descriptors for use by the SMMU.

Table 2-12 shows the SMMU architecture use of these bits.

Table 2-12 SMMU stage 2 translation table bits

PTE bits[63:60]	Name	Description	
63:62	WACFG	Write-Allocate Configuration	
		`0b00`	Use value supplied from previous stage
		`0b01`	Reserved. See *Reserved memory type and shareability attributes on page 2-62*.
		`0b10`	Write-Allocate
		`0b11`	No Write-Allocate
61:60	RACFG	Read-Allocate Configuration	
		`0b00`	Use value supplied from previous stage
		`0b01`	Reserved. See *Reserved memory type and shareability attributes on page 2-62*.
		`0b10`	Read-Allocate
		`0b11`	No Read-Allocate

The values `0b10` and `0b11` cause the allocation hint supplied as part of the transaction to be overridden with the specified value.

2.7.2 Domains

The SMMU translation process does not support the Domain feature of the ARM *Virtual Memory System Architecture* (VMSAv7) before the addition of the Large Physical Address Extension.

2.7.3 Implemented address size

For each stage of translation, the SMMU performs checks on the input address and the output address. These checks might result in a fault being generated. See *SMMU treatment of addresses on page 1-37* and *Context faults on page 3-102* for more information.

2.7.4 Global translation table entries and Security states

In common with the processor architecture, the SMMU architecture supports global and non-global memory regions.

When using the AArch32 Long-descriptor or the AArch64 translation scheme, the processor architecture can permit fetching of translation tables from Non-secure memory when the processor is in Secure state. Similarly, the SMMU architecture permits Secure context banks to access Non-secure memory during lookups.

The SMMU architecture extends this behavior so that the first level descriptor can be fetched from Non-secure memory by using the NSCFG0 and NSCFG1 fields in SMMU_CBn_TCR. This behavior is also permitted when using the Short-descriptor format.

As with the processor architecture, a Secure translation must be treated as non-global if the descriptor was fetched from Non-secure memory, regardless of the value of the nG bit.

In addition, specifically for the SMMU architecture, there is an IMPLEMENTATION DEFINED option to mark a Secure translation as non-global if the final descriptor has the NS bit set to 1, that is, it describes a mapping to Non-secure memory.

Note

- To be architecturally compliant, software mapping a Non-secure, non-global block or page into the Secure address space must set the nG bit to 1 in the translation table entry for the block or page. Similarly, for a global entry, it must set the nG bit to 0. This guarantees that the mappings work correctly across all implementations.
- ARM deprecates the behavior of an implementation forcing a non-global entry for a Secure translation with the final descriptor having the NS bit set if it was fetched from Secure memory.

The purpose of this restriction is to provide a safety measure that prevents a Secure process from accidentally polluting the memory space of another process with a global entry that maps Non-secure memory.

2.8 Hypervisor contexts (HYPC)

If the implementation supports stage 1 translation, that is, if the value of SMMU_IDR0.S1TS is 1, then:

- In an SMMUv1 implementation, and in an SMMUv2 implementation that does not include the Virtualization Host Extensions, Hypervisor context (HYPC) is available.
- In an SMMUv2 implementation that includes the Virtualization Host Extensions, when the value of SMMU_CR0.HYPMODE is 0, Hypervisor context (HYPC) is available.

Note

The value of SMMU_IDR2.E2HS is 1 if the implementation includes the Virtualization Host Extensions. For more information, see *E2H contexts (E2HC) on page 2-90*. The Virtualization Host Extensions provide an additional context, E2HC, that is an alternative to HYPC.

Translation banks for the hypervisor context are called HYPC banks, and are intended to be used by the hypervisor for translating devices that it owns and are operating on its own behalf. Hypervisor context applies when the SMMU_CBARn.HYPC bit for a bank is set to 1.

Software can configure HYPC banks as Secure or Non-secure.

Note

The register formats do not permit HYPC banks to use the stage 1 followed by stage 2 translation context.

2.8.1 Architectural effects of HYPC

In any system that includes an ARM processor, the processor contains registers that perform a similar role to certain SMMU registers. In general, the effect of HYPC in the SMMU is analogous to Hypervisor control of Non-secure access privileges, as described in:

- *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.
- *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

Table 2-13 shows the applicable SMMU registers, and the corresponding processor registers.

Table 2-13 Corresponding SMMU and ARMv8 processor hypervisor registers

SMMU register	ARMv8 processor register short name	Notes
SMMU_CBn_TTBR0	HTTBR	SMMU_CBn_TTBR1 is ignored. The SMMU_CBn_TTBR0.ASID field is ignored The following fields have no effect: • SMMU_CBn_SCTLR.AFE • SMMU_CBn_SCTLR.TEX. • SMMU_CBARn.VMID. • SMMU_CBA2Rn.VMID16.
SMMU_CBn_TCR[a]	HTCR	The SMMU_CBn_TCR[a].EAE bit is ignored and considered to be set to 1, indicating stage 1 Long-descriptor format.
SMMU_CBn_MAIRm	HMAIRn	Includes register 0 and register 1 in each case.

a. In SMMUv1, this register is SMMU_CBn_TTBCR.

Note

See Appendix A *Register Names* for more information about the ARM processor register short names, and the specific registers that apply to different Execution states.

Because there is no separation of privileged and unprivileged execution in HYPC, the stage 1 address translation registers in Table 4-1 on page 4-127 might result in different behavior. In any SMMU implementation that uses HYPC, the following registers are guaranteed to work correctly:

- SMMU_CBn_ATS1PR.
- SMMU_CBn_ATS1PW.

When targeting a HYPC bank, the global stage 1 address translation operations in Table 4-2 on page 4-130 might result in different behavior, and the following registers are guaranteed to work correctly:

- SMMU_sGATS1PR.
- SMMU_sGATS1PW.

Translation table descriptor formats for HYPC banks

The Secure EL3 translation regime uses a different permissions model to other translation regimes. For example, HYPC banks do not support the following memory attributes:

PXN Privileged execute-never bit.

AP[1] Access Permission bit.

APTable[0] Access Permissions limit for subsequent levels of lookup.

PXNTable PXN limit for subsequent levels of lookup.

UWXN Behavior for regions with unprivileged write permission.

nG Non-global bit.

In SMMUv2, HYPC banks use TTBR0 only, meaning that SMMU_CBn_TTBR1 and all fields associated with TTBR1 in SMMU_CBn_TCR and SMMU_CBn_TCR2 are ignored for HYPC banks.

2.8.2 TLB maintenance for HYPC banks

TLB entries that are allocated as a result of HYPC accesses are Hyp tagged. All Hyp tagged TLB entries are considered to belong to the same software entity and have no associated ASID or VMID.

2.8.3 HYPC and Security states

In the ARM architecture, a hypervisor exists only in Non-secure state. In the SMMU architecture, there is no requirement for HYPC banks to store and check security state, that is, there is no architectural definition of Secure or Non-secure Hyp tagged TLB entries.

It is therefore possible for a Secure HYPC bank to encounter a TLB entry allocated by a Non-secure HYPC bank, resulting in a security violation. To prevent this, Secure software must either:

- Avoid using HYPC banks, by ensuring that the SMMU_CBARn.HYPC bit is set to 0 for all Secure translation context banks.
- Prevent Non-secure software from configuring a bank as HYPC, by ensuring that:
  - The SMMU_SCR1.GASRAE bit is set to 1, to prevent Non-secure access to SMMU_CBARn.
  - A Non-secure bank is never configured with SMMU_CBARn.HYPC bit set to 1.

In addition, because software can configure a HYPC bank as Secure or Non-secure, HYPC banks must adhere to the appropriate SMMU_CBn_TCR.NSCFG0 or SMMU_CBn_TCR2.NSCFG0 settings.

When using Secure Hypervisor contexts, ARM strongly recommends that SMMU_SCR1.GASRAE is set to 1.

2.8.4 Sign-extension with HYPC banks

The SMMU_CBn_TCR2.SEP field, that indicates the Sign Extension Position, is ignored for HYPC banks. See *Sign-extension of input addresses on page 1-38* for more information.

2.8.5 Permission checking and bus marking for HYPC bank transactions

Depending on the bus standard, a hypervisor privilege tag might be included in the transaction.

If a privilege tag is not included, then the permission model for HYPC banks is independent of the bus tag. For example, a bus might only transport the concept of privileged and unprivileged, and the permission model of a HYPC bank is independent of these attributes.

Note

In SMMUv2, HYPC banks are treated as privileged on the downstream bus system, provided the downstream bus system supports such marking. It is IMPLEMENTATION DEFINED whether this attribute is recorded in SMMU_CBn_FSYNR0.PNU.

2.9 Monitor contexts (MONC)

In SMMUv2, for a Secure context bank, when the value of the corresponding SMMU_CBA2Rn.MONC bit is 1, the bank is a MONC (Secure Monitor) context bank. MONC Translation context banks are intended to be used by the Secure EL3 translation regime.

For a Secure bank, if SMMU_CBA2Rn.MONC == 1 and SMMU_CBARn.HYPC == 1, then the result of processing a transaction using the bank is UNPREDICTABLE.

For Non-secure context banks, the SMMU_CBA2Rn.MONC bit is ignored and treated as 0, for all purposes other than reading the value written to the bit. ARM recommends that Non-secure software always writes this bit as 0.

2.9.1 Translation table descriptor formats for MONC banks

The Secure EL3 translation regime uses a different permissions model to other translation regimes. For example, MONC banks do not support the following memory attributes:

PXN Privileged execute-never bit.

AP[1] Access Permission bit.

APTable[0] Access Permissions limit for subsequent levels of lookup.

PXNTable PXN limit for subsequent levels of lookup.

UWXN Behavior for regions with unprivileged write permission.

nG Non-global bit.

In SMMUv2, MONC banks use TTBR0 only, meaning that SMMU_CBn_TTBR1 and all fields associated with TTBR1 in SMMU_CBn_TCR and SMMU_CBn_TCR2 are ignored for MONC banks.

2.9.2 TLB maintenance for MONC banks

SMMUv2 provides the following global TLB invalidation operations:

- SMMU_STLBIALLM invalidates all EL3 Monitor entries.
- SMMU_STLBIVAM invalidates all EL3 Monitor entries that are associated with a 64-bit VA.
- SMMU_STLBIVALM invalidate all EL3 Monitor entries that are associated with a 64-bit VA, last level only.

Note

These operations are not required to invalidate non-MONC Secure TLB entries.

2.9.3 Sign extension with MONC banks

The SMMU_CBn_TCR2.SEP field, that indicates the Sign Extension Position, is ignored for MONC banks. See *Sign-extension of input addresses on page 1-38* for more information.

2.9.4 Permission checking and bus marking for MONC bank transactions

A transaction processed using a MONC bank is marked as privileged on the downstream bus system, if that bus system supports such marking. It is IMPLEMENTATION DEFINED whether this attribute is recorded in SMMU_CBn_FSYNR0.PNU.

2.10 E2H contexts (E2HC)

In SMMUv2, if the value of the SMMU_IDR2.E2HS bit is 1, then the implementation supports an additional context type, E2HC, that is intended to be used by an OS that is also acting as a Hypervisor. If executing on an ARM processor, this OS executes at EL2.

In an implementation that supports E2HC, SMMU_CR0.HYPMODE determines whether HYPC or E2HC contexts are available:

0 HYPC contexts are available.

1 E2HC contexts are available.

Across all context banks and all Security states, the SMMU can either use E2HC or HYPC contexts, but never both at the same time.

When available, E2HC is selected when SMMU_CBARn.TYPE is `0b01`, selecting stage 1 followed by stage 2 bypass. HYPC and E2HC occupy the same bit position in SMMU_CBARn, and those register bits are architecturally mapped to each other, but for all other purposes are logically different.

E2HC contexts provide an EL2 translation scheme that operates identically to the EL1&0 translation regime. In particular, resolution of the input address range is split between two TTBRs.

TLB entries from E2HC translations:

- Are all identified as E2H.
- Are also identified by ASID if they are non-global.
- Are not identified by Security state.
- Are required to be invalidated by the following operations, matching against all E2H, and all E2H+ASID entries:
  - SMMU_TLBIALLH.
  - SMMU_TLBIVAH.
  - SMMU_TLBIVAH64.
  - SMMU_TLBIVALH64.
  - SMMU_TLBIALLNSNH.
- Are not required to be invalidated by:
  - Secure TLBI operations.
  - SMMU_TLBIVMID.
  - SMMU_TLBIVMIDS1.

The context bank TLB invalidate operations, SMMU_CBn_TLBI*, behave the same as other non-HYPC, non-MONC bank invalidation operations, except that instead of matching a VMID they are only required to match the appropriate E2H and E2H+ASID tagged entries as appropriate.

For no other purposes than the invalidate operations mentioned in this section are E2HC context banks considered to be hypervisor contexts. All other statements in this specification about HYPC and hypervisor contexts do not apply to E2HC context.

Operation is UNPREDICTABLE if TLB entries allocated with HYPC and TLB entries with E2HC co-exist in the TLB. This means software must perform any necessary TLB invalidation to ensure that, when changing SMMU_CR0.HYPMODE, at no time can HYPC and E2HC entries exist in the TLB. ARM recommends that the process shown in Example 2-1 on page 2-91 is used when changing between using HYPC and E2HC.

Example 2-1 Changing the value of SMMU_CR0.HYPMODE from 0 to 1

1. Clear SMMU_CBARn.HYPC to 0 in all context banks.
2. Perform an SMMU_TLBIALLH invalidation.
3. Perform an SMMU_sTLBGSYNC synchronization.
4. Wait for SMMU_sTLBGSTATUS to indicate completeness. This ensures that all speculative prefetching for HYPC contexts has completed.
5. Set SMMU_CR0.HYPMODE to 1.

2.10.1 Interaction with the Security state

E2HC TLB entries are not required to be tagged by Security state and so, as with HYPC, if Secure software uses Secure E2HC transactions, then it must ensure that Non-secure software cannot allocate an E2HC context bank. In addition, the SMMU_CR0 register is Non-secure, so the Secure software must protect this register. This can be done by setting SMMU_SCR1.GASRAE to 1 to prevent any Non-secure access to the global register file.

Note

There is no explicit interaction between SMMU_SCR1.GASRAE and the E2HC or HYPC features. Software has complete responsibility for ensuring that no Security violation occurs if Secure software uses E2HC.

Chapter 3
The Fault Model

This chapter describes SMMU fault handling. It contains the following sections:

3.1 Overview of fault types

An SMMU implementation might encounter any of the following types of fault:

- A fault on a translation context bank:
  - Translation fault.
  - Permission fault.
  - External fault.
  - Unsupported upstream transaction fault.

  See *Context faults on page 3-102* for more information.

- A global fault:
  - Invalid context fault.
  - Unidentified stream fault.
  - Stream match conflict fault.
  - Unimplemented context bank fault.
  - Unimplemented context interrupt fault.
  - Configuration access fault.
  - Permission fault.
  - External fault.
  - Unsupported upstream transaction fault.
  - In SMMUv2, address size fault.

  See *Global faults on page 3-114* for more information.

The SMMU provides resources to record, process, and report these faults.

3.2 Fault-handling terminology

The following terminology applies to faults:

encounter An SMMU *encounters* a fault as a result of either:

- Processing a transaction in the SMMU.
- A transaction response returned to the SMMU by an external agent.

record An SMMU *records* a fault by:

- The appropriate Fault Status Register capturing fault status information. See:
  - *SMMU_sGFSR, Global Fault Status Register on page 9-209*.
  - *SMMU_CBn_FSR, Fault Status Register on page 16-298*.
- The corresponding Fault Address Register capturing the fault address. See:
  - *SMMU_sGFAR, Global Fault Address Register on page 9-208*.
  - *SMMU_CBn_FAR, Fault Address Register on page 16-298*.
- One or more Fault Syndrome Registers capturing additional information. See:
  - *SMMU_sGFSYNR0, Global Fault Syndrome Register 0 on page 9-211*.
  - *SMMU_sGFSYNR1, Global Fault Syndrome Register 1 on page 9-213*.
  - *SMMU_sGFSYNR2, Global Fault Syndrome Register 2 on page 9-214*.
  - *SMMU_CBFRSYNRAn, Context Bank Fault Restricted Syndrome Register A on page 10-249*.
  - *SMMU_CBn_FSYNRm, Fault Syndrome Registers on page 16-301*.

report An SMMU *reports* a fault by returning an error to the initiator of the transaction that caused the fault. Alternatively, an external system component processing a transaction issued by the SMMU might *report* a fault to the SMMU.

Support for fault reporting in the initiating device and in the interconnect between the device and the SMMU is IMPLEMENTATION DEFINED.

For exclusive access transactions, In the context of the SMMU, reporting means specifically reporting an exclusive access transaction as aborting.

interrupt An SMMU might assert an interrupt signal because of a fault.

client A *client* access is a memory transaction issued by a client device, that the SMMU is to process.

configuration A *configuration* access is a memory transaction issued by a device that accesses a register in the SMMU configuration address space.

3.3 Handling multiple memory faults

The ARM processor architecture only has to handle a single memory fault at any given time. In contrast, an SMMU implementation is a shared resource that might receive simultaneous memory access requests from multiple agents. Such concurrency means that the SMMU could encounter multiple faults in a short period of time, or could encounter a second fault before the first fault is acknowledged by software.

For a fault on a translation context bank, SMMU_CBn_FSR.MULTI is set to 1 if a fault is encountered when this register already has information about a previous fault that has not yet been acknowledged by software. Similarly, in the case of a global fault, SMMU_sGFSR.MULTI is set to 1. In both cases, the register records full details of the first fault, and only the Multiple Fault Status flag is set on a subsequent fault.

For information specific to translation context faults, see *Multiple faults on page 3-106*. For information specific to global faults, see *Multiple fault conditions on page 3-115*.

3.4 Recording memory attributes

Fault syndrome registers record the memory attributes of a transaction that caused a fault, as follows:

- For a fault on a translation context bank, SMMU_CBn_FSYNRm.
- For a global fault, SMMU_sGFSYNR0.

The SMMU architecture defines specific handling of the following memory attributes:

- *Instruction, not Data* (IND).
- *Privileged, not Unprivileged* (PNU).
- *Non-secure Attribute* (NSATTR).

The handling of these attributes is:

- If a global fault is recorded, the default input attributes IND, PNU, and NSATTR are recorded.
- If a stage 1 or a stage 2 context fault is recorded, the memory attributes IND, PNU, and NSATTR, as presented to the SMMU *after* Context determination, are recorded. This means that the values of IND, PNU, and NSATTR are possibly overridden by one of:
  - SMMU_S2CRn.INSTCFG.
  - SMMU_S2CRn.PRIVCFG.
  - SMMU_S2CRn.NSCFG, for Secure transactions.

ARM recommends that other memory attributes recorded in the Fault Syndrome Registers are handled in the same way as the IND, PNU, and NSATTR attributes, although the handling of other memory attributes is IMPLEMENTATION DEFINED.

3.5 Recording stage 1 followed by stage 2 translation faults

For a stage 1 followed by stage 2 translation, faults encountered at stage 2 are recorded either in the stage 2 context bank or in the *SMMU_sGFSR, Global Fault Status Register on page 9-209*.

The following fault types are recorded in the stage 2 context bank:

- Stage 2 permission faults.
- Stage 2 translation faults.
- External aborts.

The following fault types are recorded in the *SMMU_sGFSR, Global Fault Status Register on page 9-209*:

- Stage 1 or stage 2 invalid context fault.
- Unimplemented context bank fault.
- Unimplemented context interrupt fault.

Note

Software executing in global register space must not transfer control of a context bank to software executing at a lower privilege until the bank is fully configured. For example, this might apply to a hypervisor transferring control to a Guest OS. This can result in low privilege software affecting accesses to SMMU_sGFSR by higher privilege software.

3.5.1 Stage 2 faults for different translation contexts

In SMMUv1, there is no simple mechanism for a hypervisor to establish whether a fault in a stage 2 context bank is either:

- A result of a client transaction mapping to the stage 1 followed by stage 2 translation context, and faulting in stage 2, in which case:
  - SMMU_CBn_FAR contains the VA of the client transaction.
  - The SMMU_CBn_FSYNR0.S1PTWF bit indicates a fault on the stage 1 translation table walk.
- A result of a client transaction mapping to the stage 2 context, in which case SMMU_CBn_FAR contains the IPA of the client transaction.

This means that in SMMUv1, software must be capable of distinguishing such differences, either by:

- Using separate stage 2 banks for these different cases.
- Matching SMMU_CBFRSYNRAn with the Stream Match Register groups to determine the context.

In SMMUv2, the SMMU_sGFSYNR0.Nested bit indicates the reason for stage 2 context bank faults.

3.5.2 Behavior of stage 2 faults within stage 1 followed by stage 2 translations

This section describes behaviors associated with handling faults that occur during stage 2 translation, within a stage 1 followed by stage 2 translation that is either:

- A client transaction.
- A software-initiated address translation operation.

The behavior depends on whether stage 2 transactions are configured to terminate or to stall on a fault, as *Handling a context fault on page 3-106* describes.

Behavior when stage 2 transactions terminate on a fault

If the client transaction is terminated when a stage 2 fault occurs, no fault is recorded in stage 1, even for a stage 1 translation table walk that faults. However, depending on the setting of stage 2 SMMU_CBn_SCTLR.CFRE bit, the transaction might report an abort to the client device.

If a stage 2 fault is encountered during an address translation operation initiated in a stage 1 translation context bank, the fault is always recorded as *Address Translation Operation Terminated* (ATOT) in the stage 1 SMMU_CBn_PAR. See *Fault handling within stage 1 followed by stage 2 translations initiated in a context bank on page 4-127*.

If a transaction aborts in stage 2, the stage 2 SMMU_CBn_SCTLR.HUPCF determines the behavior of any subsequent transactions that do not abort at stage 1:

- When SMMU_CBn_SCTLR.HUPCF==0, all subsequent transactions using the stage 2 context bank terminate and record no faults in either the stage 1 or stage 2.
- When SMMU_CBn_SCTLR.HUPCF==1, the SMMU attempts to translate subsequent transactions in the stage 2 context bank. If a fault is encountered, the syndrome information is lost and a multiple fault is recorded.

This means that, if multiple stage 1 contexts share a single stage 2 context, transactions issued to a particular stage 1 context can generate a fault that affects all other stage 1 contexts. In particular, if a stage 2 context bank is configured where SMMU_CBn_SCTLR.HUPCF==0, all transactions issued by any of the connected stage 1 contexts fault silently, and no fault is recorded. However, an abort might still be reported to the client device, depending on the setting of the stage 2 SMMU_CBn_SCTLR.CFRE bit.

See *Fault behavior in virtualized context banks on page 3-123* for more information.

Note

This is different from the behavior for translations other than stage 1 followed by stage 2, where each transaction in the downstream memory system aborts independently.

Behavior when stage 2 transactions stall on a fault

If stage 2 transactions are configured to stall when a fault occurs, the hypervisor has more control than it does for transactions configured to terminate. The hypervisor is not required to ensure that all IPAs for a device have valid translations, and can:

- Make translations available on a stage 2 fault that stalls.
- Use stage 2 access flags to discover faults.
- Inject synchronous faults manually into stage 1 translation.

Independence of address translation operation faults and client transaction faults

Faults encountered during a client transaction and those encountered during an address translation operation are independent and require different SMMU resources. In general:

- Multiple independent context bank address translation operations might be in progress at any one time.
- Fault recording for global address translation operations is independent from that for client transactions.
- Fault recording for context bank address translation operations is independent from that for client transactions, except where they share a stage 2 context bank that faults.
- The SMMU records faults encountered during client transactions in SMMU_CBn_FSR.
- An address translation operation might be issued even if the SMMU records a fault in SMMU_CBn_FSR.

Note

If a context bank is configured incorrectly, this could permit inappropriate Guest OS access to SMMU_sGFSR. This could happen if, for example, an invalid stage 2 context identifier exists within a stage 1 followed by stage 2 transaction.

3.6 Fault interrupts

An SMMU can communicate faults to software using the following types of interrupt:

- Global interrupt.
- Context interrupt.

3.6.1 Global interrupts

An SMMU implements at least the following logical Global interrupts:

- SMMU_NsgCfgIrpt.
- SMMU_NSgIrpt.

SMMU_NsgCfgIrpt is used when a configuration access fault is recorded. SMMU_NSgIrpt is used when any other type of global fault is recorded.

In an implementation that supports two Security states, the SMMU also implements SMMU_gCfgIrpt and SMMU_gIrpt. These are the Secure equivalents of SMMU_NsgCfgIrpt and SMMU_NSgIrpt respectively.

3.6.2 Context interrupts

This section describes the differences in context interrupt behavior between SMMUv1 and SMMUv2.

Context interrupts in SMMUv1

In SMMUv1, a Context interrupt is raised for a fault on a translation context bank. SMMU_IDR0.NUMIRPT specifies the IMPLEMENTATION DEFINED number of Context interrupts supported by the SMMU. SMMU_CBARn.IRPTNDX specifies the interrupt number to assert in the event of a translation context bank fault.

ARM recommends that:

- SMMU_CBARn.IRPTNDX is software configured in the range specified by SMMU_IDR0.NUMIRPT.
- If Secure software gives Non-secure software at least one translation context bank, the Secure software provides the Non-secure software with at least one Context interrupt.

An Unimplemented context interrupt fault occurs when all of the following conditions are true:

- SMMU_CBARn.IRPTNDX is configured outside the range reported by SMMU_IDR0.NUMIRPT.
- Stage 1 SMMU_CBn_SCTLR.CFIE or stage 2 SMMU_CBn_SCTLR.CFIE is 1.
- A context fault causes the assertion of a Context interrupt.

Note

It is IMPLEMENTATION DEFINED whether SMMU_CBARn.IRPTNDX is fully implemented with writable storage. A portion of this field is permitted to behave as RAZ/WI, because the implementation might not provide sufficient interrupts for some of the upper bits of SMMU_CBARn.IRPTNDX to be relevant. The implemented range of this field is the same for all SMMU_CBARn registers in an implementation.

In an implementation that supports two Security states, SMMU_SCR1.NSNUMIRPTO allocates Context interrupts for use by Secure software.

An SMMU_CBARn register allocated for use by Secure software must only specify a Context interrupt allocated for use by Secure software. Otherwise, UNPREDICTABLE behavior results.

An SMMU_CBARn register not allocated for use by Secure software can only specify a Context interrupt not allocated for use by Secure software. Otherwise, an Unimplemented context interrupt fault could occur.

Context interrupts in SMMUv2

SMMUv2 supports one of the following interrupt mechanisms:

- When SMMU_IDR0.NUMIRPT==`0b00000001`, each context bank has a dedicated interrupt pin.

- When SMMU_IDR0.NUMIRPT is greater than 0b00000001, this specifies the number of interrupts provided. Each interrupt can be assigned to any context bank, as for SMMUv1.

When each context bank has a dedicated interrupt pin, there is no requirement for software to specify the interrupt number, and:

- The SMMU_IDR0.NUMIRPT field is always 0b00000001.
- The SMMU_CBARn.IRPTNDX field is 0b00000000.
- The SMMU_SCR1.NSNUMIRPTO field is always 0b00000001.

3.6.3 Interrupt assertion

An SMMU interrupt is asserted in response to a fault recorded in the Fault Status Register, providing interrupts are enabled.

An interrupt is asserted regardless of whether the Fault Status Register is set by a fault or by a nonzero value written to SMMU_sGFSRRESTORE or SMMU_CBn_FSRRESTORE.

With the exception of the SS and Format bits, a value of 1 written to any non-reserved bit in SMMU_CBn_FSR or SMMU_sGFSR clears that bit. A value of 0 written to any of these bits leaves the bit unchanged.

For global Secure or Non-secure Configuration Access fault interrupts, the interrupt is asserted whenever enabled by SMMU_sCR0.GCFGIE, and the corresponding SMMU_sGFSR.CAF bit is 1.

For global Secure or Non-secure fault interrupts that are other than Configuration Access faults, the interrupt is asserted whenever enabled by SMMU_sCR0.GFIE, and any bit other than the CAF bit is set to 1.

In SMMUv1, for context interrupts, multiple context banks can be configured to use a particular interrupt. If no context bank is configured to use a particular interrupt, then it is deasserted. An interrupt is asserted only when any context bank assigned to it is asserting an interrupt.

In SMMUv2, an implementation may either:

- Have the same behavior as SMMUv1 and allow context bank interrupts to be programmed to any interrupt.
- Have a non-assignable context interrupt per context bank.

See *Context interrupts on page 3-100* for more information.

A context bank only asserts an interrupt when enabled by SMMU_CBn_SCTLR.CFIE and SMMU_CBn_FSR indicate a fault.

It is IMPLEMENTATION DEFINED as to whether the SMMU_CBn_FSR.SS bit is considered a fault for the purposes of generating an interrupt. All other bits, with the exception of SMMU_CBn_FSR.Format field, if nonzero, are considered to indicate a fault. For example, if a transaction stalled because of a permission fault, then SMMU_CBn_FSR.{SS, PF} would be one. If software cleared the PF bit, while the transaction remains stalled, then it is IMPLEMENTATION DEFINED if the interrupt is still asserted.

3.7 Context faults

A context fault is associated with a particular translation context bank. The following types of context fault exist:

- Translation fault.
- Permission fault.
- External fault.
- In SMMUv2, address size fault.

A translation fault occurs if the SMMU does not obtain a translation for a transaction, or in SMMUv1, if an input address is out of range.

A permission fault occurs if the SMMU retrieves a translation for a transaction, but the transaction has insufficient privileges to reach completion.

An external fault occurs if an external abort is reported to the SMMU during transaction processing.

In SMMUv2:

- An address size fault can occur when an output address contains an unexpected number of bits.
- An unsupported upstream transaction fault can occur for certain client transactions.

3.7.1 *Secure Instruction Fetch* (SIF) permission faults

In an implementation that supports two Security states, Secure instruction fetches can result in permission faults when the *Secure Instruction Fetch* (SIF) bit in SMMU_SCR1 is set to 1. A permission fault results when a Secure domain client access attempts to exit as a Non-secure instruction *after* any transformation indicated by the SMMU_S2CRn.INSTCFG bit has been applied.

For transactions associated with a particular translation context bank, a SIF permission fault is recorded using the SMMU_CBn_FSR.PF bit when all of the following apply:

- The SMMU_SCR1.SIF bit is set to 1.
- The transaction has been routed to a Secure context bank.
- The transaction is classified as Instruction, *after* applying any SMMU_S2CRn.INSTCFG transformation.
- The outgoing transaction is Non-secure.

If the fault is associated with a translation table walk, the last level of address lookup is recorded in the SMMU_CBn_FSYNR0.PLVL field. If no translation table was performed, the value written to PLVL is:

- 0, if using AArch64.
- 1, if using AArch32.

This check is also applied to transactions that bypass the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56* for more information.

Note

For client transactions that use stage 1 or stage 2 context resources, the instruction fetch configuration attribute specified by the SMMU_S2CRn.INSTCFG bit can override the incoming memory attributes. See *Stream-to-Context Register, SMMU_S2CRn on page 2-55*. When referring to instruction fetches in this document, it is assumed that any such overrides are accounted for.

3.7.2 Recording a context fault

The registers that record information about a transaction causing a fault on a translation context bank are:

- SMMU_CBn_FSR.
- SMMU_CBn_FAR.
- SMMU_CBn_FSYNRm.

SMMU_CBFRSYNRAn records additional fault syndrome details about the fault, where *n* is the index into the translation context bank.

Two-stage translation faults

A context fault is only recorded for one stage of translation. This means that in an implementation supporting two translation stages:

- If no stage 2 fault is encountered, a context fault that occurs is recorded for the stage 1 translation context.
- If a stage 2 fault is encountered, the fault is recorded for the stage 2 translation context.

External faults

If an external fault is reported to the SMMU in response to a fetch issued as part of a translation table walk, the fault is recorded synchronously in the translation context bank. If any other external fault is reported to the SMMU in response to a transaction associated with a translation context bank, it is IMPLEMENTATION DEFINED whether the fault is:

- Recorded synchronously.
- Recorded asynchronously.
- Not recorded.

Synchronous recording of an external fault updates the following registers:

- SMMU_CBn_FSR.
- SMMU_CBn_FAR.
- SMMU_CBn_FSYNRm.
- SMMU_CBFRSYNRAn.

Asynchronous recording of an external fault updates the following registers:

- SMMU_CBn_FSR.
- SMMU_CB*n*_FSYNR0.AFR is set to 1, as *SMMU_CBn_FSYNRm, Fault Syndrome Registers on page 16-301* describes.
- For an external fault on a translation table walk, SMMU_CB*n*_FSYNR0.PTWF is set to 1. Otherwise, it is cleared to 0, as *SMMU_CBn_FSYNRm, Fault Syndrome Registers on page 16-301* describes.
- Other fields in the following registers are UNK/SBZP:
  - SMMU_CBn_FSYNRm.
  - SMMU_CBn_FAR.

For more information about external faults, see *External faults on page 3-120*.

Unsupported upstream transaction faults

Depending on the upstream bus system, certain client transactions might result in an unsupported upstream transaction fault. These can either be context faults or global faults.

Where the client device is associated with a context bank, the SMMU records unsupported upstream transactions as context faults. To record an unsupported upstream transaction fault, the SMMU:

- Updates the appropriate SMMU_CBn_FSR.
- Aborts the transaction if the SMMU_CBn_SCTLR.CFRE bit is set to 1.

The architecture only allows UUT faults to be generated at the following points:

- After SSD but before context bank determination, and will be reported to the global fault status register group. For example, when the upstream bus system can represent transactions that cannot be expressed on the downstream bus system.
- After context bank determination but before the page tables or the TLBs are examined, and will be reported to the appropriate context bank fault status register group. For example, where the SMMU receives more than 49 bits of VA and cannot suitably sign-compress the additional bits.
- After translation and after permission fault and SIF fault checks, and will be reported to the appropriate context bank fault status register group. For example, when the translated attributes of the transaction are incompatible with the downstream bus system.

Note

It is IMPLEMENTATION DEFINED whether the SMMU records a particular unsupported upstream transaction fault.

Translation faults

ARMv7 provides no mechanism for representing a fault where the incoming address is greater than the permitted input address. Therefore, in SMMUv1, this condition results in a translation fault. Because ARMv8 supports all input address sizes, SMMUv2 provides no input address checks when SCTLR_CBn_SCTLR.M== 0.

The following tables specify the input address check conditions that result in a translation fault:

- Table 3-1, for stage 1 translations
- Table 3-2, for stage 2 translations

Table 3-1 Conditions where a stage 1 input address generates a translation fault

SMMU_CBA2Rn.VA64	Value of SMMU_CBn_SCTLR.M	
	0	1
0	VA > 32 bits.	Either: • VA > 32 bits. • VA out of the range specified by SMMU_CBn_TCR.T0SZ and SMMU_CBn_TCR.T1SZ.
1	-	Either: • Bits above the bit position specified by SMMU_CBn_TCR2.SEP are nonzero. • The VA after sign extension[a] is out of range specified by SMMU_CBn_TCR.T0SZ or SMMU_CBn_TCR.T1SZ.

a. See *Sign-extension of input addresses on page 1-38* for more information.

Table 3-2 Conditions where a stage 2 input address generates a translation fault

SMMU_CBA2Rn.VA64	Value of SMMU_CBn_SCTLR.M	
	0	1
0	IPA > *IPA size*.	Either: • *IPA > IPA size.* • IPA greater than the size indicated by SMMU_CBn_TCR.T0SZ.
1	-	Either: • IPA > *IPA size.* • IPA greater than the size indicated by SMMU_CBn_TCR.T0SZ.

See *SMMU address size parameters on page 1-36* for definitions of the terms used in Table 3-1 and Table 3-2.

Address size faults

SMMUv2 supports address size faults. The SMMU checks the output address size for each stage of translation, and reports an address size fault if an address contains more bits than it expects. The output address that the SMMU checks:

- Can be either an IPA or a PA, for stage 1 translation.
- Is always a PA, for stage 2 translation.

For transactions that requires only stage 2 translation, the SMMU might also check the input address.

The following tables specify the output address check conditions that result in an address size fault:

- Table 3-3, for stage 1 translations.
- Table 3-4, for stage 2 translations.

Table 3-3 Conditions where a stage 1 output address generates an address size fault

SMMU_CBA2Rn.VA64	Value of SMMU_CBn_SCTLR.M	
	0	1
0	IPA is greater than the minimum of 2^{40} and the size indicated in SMMU_IDR2.IAS[a]	IPA is greater than the minimum of 2^{40} and the size indicated in SMMU_IDR2.IAS[a]
1	IPA is greater than the *effective IPA size*	IPA is greater than the minimum of the *effective IPA size* and *PA size*

a. This differs from the equivalent condition in the ARM processor architecture.

Table 3-4 Conditions where a stage 2 output address generates an address size fault

SMMU_CBA2Rn.VA64	Value of SMMU_CBn_SCTLR.M	
	0	1
0	PA is greater than the minimum of 2^{40} and the size indicated in SMMU_IDR2.OAS[a]	PA is greater than the minimum of 2^{40} and the size indicated in SMMU_IDR2.OAS[a]
1	PA is greater than the *output address size*	PA is greater than the minimum of *output address size* and *PA size*

a. This differs from the equivalent condition in the ARM processor architecture.

See *SMMU address size parameters on page 1-36* for definitions of the terms in Table 3-3 and Table 3-4.

For Stage 1 contexts, the conditions in Table 3-3 apply to an IPA that is either:

- An IPA produced by a stage 1 translation.
- An IPA that is the result of a translation table walk.
- The 48-bit address in SMMU_CBn_TTBR0 or SMMU_CBn_TTBR1.

For Stage 2 contexts, the conditions in Table 3-4 apply to a PA that is either:

- A PA produced by a stage 1 or stage 2 translation.
- A PA that is the result of a translation table walk.
- The 48-bit address in SMMU_CBn_TTBR0.

Multiple faults

SMMU_CBn_FSR.MULTI indicates the presence of multiple outstanding faults.

If a fault is encountered when all of the fields in SMMU_CBn_FSR are zero, including SMMU_CBn_FSR.MULTI, the following registers record full details of the fault:

- SMMU_CBn_FSR.
- SMMU_CBn_FAR.
- SMMU_CBn_FSYNRm.

If a fault is encountered when SMMU_CBn_FSR is nonzero:

- The SMMU_CBn_FSR.MULTI bit is set to 1.
- Details of the fault are *not* recorded.

Context Bank Fault Restricted Syndrome Register, SMMU_CBFRSYNRA*n*

There is an SMMU_CBFRSYNRAn register for each translation context bank. *n*, the translation context bank index, denotes the matching SMMU_CBFRSYNRAn register in the range 0-127. The register provides information about the fault recorded in the SMMU_CBn_FSR register of a stage 1 or a stage 2 translation context bank.

The information recorded in SMMU_CBFRSYNRAn could present a virtualization hole if the register were to reside in the translation context bank address space, potentially accessible by a Guest OS. Therefore, each SMMU_CBFRSYNRAn register is in the global address space. For information about processor virtualization, including what is meant by a Guest OS in this context, see the *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

The SMMU_CBFRSYNRAn registers are organized as a table. Because a SMMU_CBFRSYNRAn register exists for each translation context bank, an SMMU implementation must implement the same number of SMMU_CBFRSYNRAn registers as the number of translation context banks. This means that the number of SMMU_CBFRSYNRAn registers implemented must match the number of translation context bank table entries.

A configuration access to an unimplemented SMMU_CBFRSYNRAn register results in either of the following IMPLEMENTATION DEFINED behaviors:

- The access is RAZ/WI.
- A configuration access fault.

For more information about configuration access faults, see *Configuration access on page 3-119*.

In an implementation that supports two Security states, SMMU_SCR1.NSNUMCBO reduces the number of translation context banks, and therefore the number of SMMU_CBFRSYNRAn registers, that are visible to Non-secure software.

SMMU_IDR1.NUMCB indicates the number of implemented SMMU_CBFRSYNRAn registers.

3.7.3 Handling a context fault

An SMMU handles a context fault by either terminating or stalling the transaction that caused the fault.

If the SMMU terminates the fault, the SMMU does not perform the final memory access, and client transactions behave as RAZ/WI. Depending on the value of the stage 1 or stage 2 SMMU_CBn_SCTLR.CFRE field, the SMMU reports the fault to the initiator of the faulty transaction. See *Context interrupts on page 3-100* for more information.

If the SMMU stalls the fault, software can subsequently write to SMMU_CBn_RESUME to retry or terminate the stalled transaction. The SMMU_CBn_FSR.SS field is cleared as a result of invoking the operation to resume the transaction.

The stage 1 SMMU_CBn_SCTLR.CFCFG and stage 2 SMMU_CBn_SCTLR.CFCFG registers handle context faults on a per context basis.

It is IMPLEMENTATION DEFINED whether the SMMU supports the stall mode of operation.

The Stall fault model

If a fault is encountered by the SMMU when the Stall fault model is enabled, the SMMU stalls the transaction that caused the fault, pending action to rectify the cause.

ARM recommends that the Stall fault model generally be used with fault interrupting. This is so that a supervisory agent can be signaled to rectify the cause and restart the transaction.

Stage 1 SMMU_CBn_SCTLR.CFCFG and stage 2 SMMU_CBn_SCTLR.CFCFG enable the Stall fault model, per translation context.

It is IMPLEMENTATION DEFINED whether an SMMU implementation includes the Stall fault model. If an implementation does not include the Stall fault model, stage 1 SMMU_CBn_SCTLR.CFCFG and stage 2 SMMU_CBn_SCTLR.CFCFG each has a fixed value and are RAZ/WI.

Stage 1 SMMU_CBn_SCTLR.HUPCF and stage 2 SMMU_CBn_SCTLR.HUPCF provide more configuration options for the Stall fault model, as follows:

- If stage 1 SMMU_CBn_SCTLR.HUPCF or stage 2 SMMU_CBn_SCTLR.HUPCF is 0 and a fault occurs, no more transactions for that translation context bank are processed until the fault is cleared.
- If stage 1 SMMU_CBn_SCTLR.HUPCF or stage 2 SMMU_CBn_SCTLR are 1 and a fault occurs, more transactions can be processed for that translation context bank before the original fault is cleared. See *Hit-under-fault on page 3-108* for more information.

  It is IMPLEMENTATION DEFINED whether an SMMU implementation includes functionality for processing a transaction under an existing fault. If the implementation does not provide this functionality:

  — The behavior is identical to stage 1 SMMU_CBn_SCTLR.HUPCF or stage 2 SMMU_CBn_SCTLR.HUPCF being 0.

  — Stage 1 SMMU_CBn_SCTLR.HUPCF and stage 2 SMMU_CBn_SCTLR.HUPCF remain writable.

  The number of transactions processed after the original faulty transaction is IMPLEMENTATION DEFINED.

  The number of subsequent transactions that can raise a fault before no more transactions are processed for that translation context bank until the original fault condition is cleared, is also IMPLEMENTATION DEFINED.

Depending on the SMMU implementation, system topology, and devices connected to the SMMU, it might not be possible to guarantee that a transaction in one translation context is unaffected by a stalled transaction in another context. Therefore, use of the Stall fault model in a translation context might not be appropriate.

SMMU_sCR0.STALLD can be configured to globally disable the Stall fault model. This forces each stage 1 SMMU_CBn_SCTLR.CFCFG or stage 2 SMMU_CBn_SCTLR.CFCFG to be RAZ/WI.

In an implementation that supports two Security states:

- SMMU_CR0.STALLD must apply to Non-secure contexts banks, and optionally, can apply to Secure translation context banks.
- SMMU_SCR0.STALLD is an override bit that has equivalent behavior to SMMU_CR0.STALLD.

Note

ARM expects that SMMU_sCR0.STALLD will be configured as part of the reset initialization process.

In response to a stalled context fault, if the SMMU is configured to raise an interrupt, supervisory software typically performs one of the following actions:

- Fixes the faulty translation and retries processing the stalled transaction by writing the appropriate value to SMMU_CBn_RESUME. Fixing the faulty transaction might involve updating translation tables and TLB maintenance.
- Terminates the stalled transaction by writing the appropriate value to SMMU_CBn_RESUME. A transaction terminated in this way returns no data for reads, and writes are ignored.

  Depending on the setting of stage 1 SMMU_CBn_SCTLR.CFRE or stage 2 SMMU_CBn_SCTLR.CFRE, the SMMU can report an abort to the initiator of the transaction. See *Context interrupts on page 3-100*.

Fault recording does not occur for a transaction terminated by the SMMU_CBn_RESUME operation. This is because the fault is logged when the transaction first stalls.

In SMMUv2, if a context bank has a stalled transaction, indicated by SMMU_CBn_FSR.SS==1, when the corresponding SMMU_CBARn or SMMU_CBA2Rn is written, an implementation might retry the stalled transaction immediately after SMMU_CBARn or SMMU_CBA2Rn is updated. That is, the implementation might exhibit behavior corresponding to SMMU_CBn_RESUME.TnR==0. An implementation might also retry a transaction in all context banks affected by a change to SMMU_SCR1.NSNUMCBO.

Hit-under-fault

Optionally, an SMMU can continue to process a transaction before the condition causing an existing fault is rectified. For a translation context, Hit-under-fault behavior, if enabled, means that any subsequent transaction mapped to that context is processed, regardless of whether an outstanding fault is recorded for that context.

The following types of behavior apply:

- If termination behavior is selected for a translation context and a fault is active for that context, each subsequent transaction is processed separately. If a fault is raised on a subsequent transaction, it terminates and SMMU_CBn_FSR records a multiple fault condition.
- If stall behavior is selected for a translation context and a fault is active for that context, each subsequent transaction is processed separately. If a fault is raised on a subsequent transaction, that transaction waits, no fault status information is recorded on it, and it is retried after the original fault condition clears.

If Hit-under-fault is not enabled, any subsequent transaction mapped to that translation context is processed the same as the original faulty transaction, as follows:

- If termination behavior is selected for a translation context and a fault is active for that context, each subsequent transaction terminates. SMMU_CBn_FSR does *not* record any fault condition for any subsequent transaction beyond the original transaction. The termination of a subsequent transaction is regarded as a side-effect of the original fault.
- If stall behavior is selected for a translation context and a fault is active for that context, any subsequent transaction stalls, and only resumes after the original fault condition is cleared.

Regardless of whether Hit-under-fault is enabled, whether an active fault is recorded depends on the setting of the stage 1 or stage 2 SMMU_CBn_SCTLR.CFRE bit, as *Reporting a context fault on page 3-113* describes.

Stage 1 SMMU_CBn_SCTLR.HUPCF and stage 2 SMMU_CBn_SCTLR.HUPCF adjust the Hit-under-fault model.

Note

For stage 2 translations, when SMMU_CBn_SCTLR.HUPCF is set to 0, all subsequent transactions that use stage 1 translation are stalled or terminated, including when a matching TLB entry is located.

Context fault model pseudocode

To perform a translation request, the SMMU:

1. Determines whether any of the context banks that the translation uses are faulting,
2. For any faulting context bank, determines whether this transaction is required to perform an action.
3. Performs the translation operation.
4. If the translation process fails, re-examines the associated Fault Status Register groups to determine:
   - Whether the fault is to be recorded.
   - How the fault is to be recorded.
   - Whether the SMMU is required to report to the upstream client device.

In general, the SMMU handles transactions in an unknown order. However, any bus ordering requirements must be satisfied. For example, an SMMU that receives transactions A, B, and C from the same device can process these requests in any order, unless the underlying bus system specifies a particular order. For example, in an AMBA system, barriers affect the ordering of processing of transactions and this must be preserved by the SMMU.

Note

The SMMU might prefetch any TLB entry for any reason at any time, subject to TLB prefetch behavior rules described in *Updating the state of a context bank on page 5-134*. This means that sometimes step 3 might be speculatively performed before or during steps 1 and 2.

The following pseudocode describes how the SMMU processes translation requests and records faults, including the interactions between stall, terminate, and Hit-under-fault behaviors.

```
///
/// Conceptually transactions are received into request buffers that
/// run the following state machine once they have determined a valid
/// context bank
///
/// Note: request buffers containing data transactions can exit from a "wait"
/// in a different order to the order transactions were received or entered a "wait"
///
/// Note: the restart of these buffers must be consistent with ARM ARM sections:
/// - A3.5 Memory types and attributes and the memory order model
/// - B2.2.9  Ordering of cache and branch predictor maintenance operations
///
/// Note: for barrier operations, request buffers must restart in an order
/// consistent with the ARM ARM.  For example earlier transactions from the
/// same master that issued the barrier are restarted before the barrier and
/// later transactions are restarted after the barrier.  For a distributed
/// SMMU implementation where separate TLBs are associated with individual bus
/// segments, then the request buffers associate with each TLB are unordered
/// with respect to the TLBs associated with other bus segments because each bus
/// segment can be assumed to belong to different sets of masters.
///
/// Note: this pseudocode executes atomically between "wait" calls so there
/// are no race conditions on reading and writing registers unless there
/// is an intervening "wait"
///
/// This function returns true if an abort whether be generated to the
/// upstream client device and whether the transaction can be sent downstream
///
/// The intent is that this pseudocode describes a simple to implement
/// hardware state machine and ensure a programmers model that is easy
/// to reason about

enum RequestAction
{
    OkaySendDownStream,
    AbortReportToClient,
    AbortDoNotReportToClient,
    GotoStart,                // Never returned from RequestBufferStateMachine
    CheckPassed               // Never returned from RequestBufferStateMachine
}

RequestAction RequestBufferStateMachine()
{
    bool first = true;

    // A variable that holds any data that can be reused between different
    // invocations of DoTranslation()
    IMP_DEF_TYPE restartData = new IMP_DEF_TYPE();

    start:
        // See the section on "Enabling Global stall mode"
        if (GSErequiresStalling() && any bank is stalled && imp_def_option_1)
```

```
        WaitForWriteToResume(bankThatIsStalled);
        goto start;

    RequestAction action = CheckForFaultsInBank( initial_bank );

    if (action == GotoStart)
        goto start;
    else if (action != CheckPassed)
        return action;

    // NOTE: even if a transaction would abort in the stage 1 bank
    // without using stage 2, an existing fault in stage 2 will cause
    // this transaction to stall or terminate.  Examples of this are:
    // -- Translation Fault due to T*SZ or EPD0/1
    // -- Address Size Fault due to TTBR0/1
    // This means such a transaction might stall or terminate immediately
    // without recording in Stage 1.  In the case where such a transaction
    // stalls, after the existing stage 2 fault has been cleared, the
    // transaction will restart and (in the normal case) record the stage 1
    // fault.

    if (this is stage 1 and stage 2 exists)
    {
        action = CheckForFaultsInBank( stage2_bank );
        if (action == GotoStart)
            goto start;
        else if (action != CheckPassed)
            return action;
    }

    // At this point we have decided that this transaction can go ahead
    // and be translated. We do NOT resample FSR, CFCFG or HUPCF for this stage
    // of translation.
    //
    // NOTE: DoTranslation() does not read or write any of the fault status
    //       register groups.  Therefore if this translation context bank or
    //       any associated stage 2 context bank becomes faulting due to a
    //       different transaction, it does not affect the behaviour of
    //       DoTranslation().  If DoTranslation() itself encounters a
    //       fault condition, it returns false and the fault recording is as
    //       below.  This allows a distributed page table walk mechanism.
    bool translationOK = DoTranslation(first, &restartData);

    first = false;

    if (translationOK)
    {
        // Does NOT resample FSR, HUPCF or CFCFG as the translation was okay
        // Note: if HUPCF is zero and a separate request has recorded a fault
        // in FSR then we still go downstream.  This means a programmer
        // cannot rely on reading FSR != 0 and HUPCF == 0 meaning that no
        // new downstream transactions can be started.
        return OkaySendDownstream;
    }

    // We have a fault and must resample FSR, CFCFG and HUPCF
    if (FaultSetIn(faultingBank.SMMU_CBn_FSR))
    {
        // A fault is already recorded in the FSR.  Note: we do not sample
        // CFCFG here.

        if (faultingBank.SMMU_CBn_FSR.SS == 1)
        {
            // There is a stalled request awaiting a resume however
            // CFCFG might have change and might currently be Terminate
            // but we still obey the stall model active when the fault
            // was initially recorded
            WaitForWriteToResume(faultingBank);
```

```
                goto start;  // We must resample CFCFG
            }

            // CFCFG is either Terminate or has been changed to Stall but this
            // change will not take effect until the FSR has been cleared.

            if (faultingBank.SMMU_CBn_SCTLR.HUPCF == 0)
            {
                // FSR is not updated
                // Return abort to upstream device depending on value
                // of SMMU_CBn_SCTLR.CFRE
                return ClientAbort(faultingBank);
            }
            else
            {
                // There is already a fault recorded in the FSR
                // so set MULTI
                faultingBank.SMMU_CBn_FSR.MULTI= 1;
                // Return abort to upstream device depending on value
                // of SMMU_CBn_SCTLR.CFRE
                return ClientAbort(faultingBank);
            }
        }
        else
        {
            // No prior fault recorded in FSR

            if ( InTerminateMode( faultingBank ) )
            {
                // Terminate model
                RecordFaultInFSR(faultingBank, .SS = 0);
                // Return abort to upstream device depending on value
                // of SMMU_CBn_SCTLR.CFRE
                return ClientAbort(faultingBank);
            }
            else
            {
                // Stall model
                RecordFaultInFSR(faultingBank, .SS = 1);
                bool terminate = WaitForWriteToResume(faultingBank);

                if (terminate)
                {
                    // Return abort to upstream device depending on value
                    // of SMMU_CBn_SCTLR.CFRE
                    return ClientAbort(faultingBank);
                }
                goto start;  // We must resample CFCFG
            }
        }

    // Unreachable
    Assert(0);

    /// Local methods

    RequestAction CheckForFaultsInBank( context_bank_t bank )
    {
        if (FaultSetIn(bank.SMMU_CBn_FSR))
        {
            if ( InTerminateMode( bank ) )
            {
                // Terminate model
                if (bank.SMMU_CBn_FSR.SS == 1
                 && bank.SMMU_CBn_SCTLR.HUPCF == 0)
                {
                    // If previous transactions are stalled (by the Stall Model)
                    // then CFCFG has been changed dynamically and HUPCF is
```

```
                    // not allowed so wait for these previous transactions
                    // to be resumed or terminated
                    // by a write to SMMU_CBn_RESUME (this clears SS)
                    WaitForWriteToResume(bank);
                    return GotoStart;  // We need to re-sample CFCFG
                }

                if (bank.SMMU_CBn_SCTLR.HUPCF == 0)
                {
                    // FSR is not updated
                    // Return abort to upstream device depending on value
                    // of the current bank's SMMU_CBn_SCTLR.CFRE
                    return ClientAbort(bank);
                }

                ; // Fall through and allow translation
            }
            else
            {
                // Stall model
                if (bank.SMMU_CBn_SCTLR.HUPCF == 0)
                {
                    // In the stall model, we wait for the FSR
                    // to be free if HUPCF is zero and there is a
                    // stalled transaction
                    if (bank.SMMU_CBn_FSR.SS == 1)
                    {
                        WaitForWriteToResume(bank);
                        return GotoStart;
                    }
                    else
                    {
                        // If SS is zero but other bits are nonzero, a prior fault
                        // has terminated but CFCFG has since been changed
                        //
                        // Obey the new mode (which is Terminate) and abort
                        // the transaction.
                        return ClientAbort(bank);
                    }
                }

                // There is a fault recorded in the FSR and SS may or may not
                // be set.  If SS is zero, then if this transaction faults,
                // FSR.MULTI might be set (see above).

                ; // Fall through and allow translation
            }
        }

        return CheckPassed;
    }

    RequestAction ClientAbort(context_bank_t bank)
    {
        if (bank.SMMU_CBn_SCTLR.CFRE == 1)
        {
            return AbortReportToClient;
        }

        return AbortDoNotReportToClient;
    }

    bool InTerminateMode(context_bank_t bank)
    {
        return bank.SMMU_CBn_SCTLR.CFCFG == 0
               || (bank is secure
                   && (SMMU_SCR0.STALLD == 1
                      || (SMMU_CR0.STALLD && imp_def_option_0)))
```

```
            || (bank is non secure
                && (SMMU_CR0.STALLD == 1 || SMMU_SCR0.STALLD == 1))
    }

    bool FaultSetIn(bits(32) fsr)
    {
        // Mask out the Format field in bits [10:9]
        return (fsr & ~0x00000600) != 0;
    }
}
```

3.7.4 Reporting a context fault

The SMMU can be configured to report a context fault. The value of SMMU_CBn_SCTLR.CFRE in the stage of translation that encounters the fault indicates whether an abort is reported to the initiator of a terminated transaction.

3.7.5 Enabling Global stall mode

The SMMU might give the ability to globally stall the processing of each translation that arrives after a stall fault is raised in a translation context bank. The global stall applies to all transactions arriving at the SMMU, regardless of the translation context they map to.

SMMU_sCR0.GSE enables global stalling.

It is IMPLEMENTATION DEFINED whether global stall is supported:

- In an implementation that does not support global stalling, SMMU_sCR0.GSE is RAZ/WI.
- In an implementation that supports global stalling, SMMU_sCR0.STALLD can disable global stalling. When SMMU_sCR0.STALLD is 1, SMMU_sCR0.GSE is RAZ/WI.

In an implementation that supports two Security states:

- SMMU_CR0.GSE must apply to Non-secure translation context banks, and an implementation is permitted to apply the setting of SMMU_CR0.GSE to transactions processed by Secure translation context banks.
- SMMU_SCR0.GSE provides functionality that is equivalent to the functionality provided by SMMU_CR0.GSE.
- SMMU_CR0.STALLD must apply to SMMU_CR0.GSE, and optionally, can apply to SMMU_SCR0.GSE. Equivalent requirements exist for SMMU_SCR0.STALLD.

3.8 Global faults

A global fault is a fault that occurs when either:

- A transaction being processed by the SMMU has no associated translation context bank.
- A translation context bank is not the appropriate place to record the fault.

The possible causes of a global fault are:

- An Invalid context fault is generated because a fault context is selected. This means either:
  - The Stream mapping register group that matches the transaction has SMMU_S2CRn.TYPE == `0b10`, that is, the fault context.
  - The SMMU_CBARn register of the initial context specified by the Stream mapping process has SMMU_CBARn.TYPE == `0b10`, that is, stage 1 with stage 2 faulting.

  In SMMUv1, if the context representing the stage 2 of a stage 1 followed by stage 2 translation is not marked as SMMU_CBARn.TYPE == `0b00`, that is, stage 2 only, then the result is UNPREDICTABLE. In SMMUv2, this case produces an Invalid context fault.

- An Invalid context fault is generated because either:
  - SMMU_CBARn.TYPE selects an unsupported context.
  - The value of SMMU_CBARn.TYPE for a Secure bank context is not `0b01`, that is stage 1 with stage 2 bypass.

- An Invalid context fault is generated because the stage 2 context indicated by a stage 1 followed by stage 2 configuration meets one or more of the following conditions:
  1. Does not exist.
  2. Is Secure.
  3. Has SMMU_CBA2Rn.VA64 == 0 when the stage 1 context has SMMU_CBA2Rn.VA64 == 1. That is, stage 1 is a 64-bit translation regime and stage 2 is a 32-bit translation regime.
  4. In SMMUv2, is not marked as SMMU_CBARn = `0b00`, to indicate a stage 2 context.

     Note

     In SMMUv1, this condition makes the result UNPREDICTABLE.

- An Unidentified stream fault, where both:
  - No match for the transaction is found in the Stream matching registers.
  - The value of SMMU_sCR0.USFCFG is 1. This enables generation of the Unidentified stream fault.

- A stream match conflict fault, where both:
  - Multiple matches for transaction are detected in the Stream matching registers.
  - The value of SMMU_sCR0.SMCFCFG is 1. This enables generation of the Stream match conflict fault.

- An Unimplemented context bank fault, where the transaction maps to an unimplemented translation context. For example, in an implementation that does not support stage 2 translation as part of a stage 1 followed by stage 2 translation, an attempt to use a stage 2 translation context bank might generate an Unimplemented context bank fault.

- An Unimplemented context interrupt fault, where a transaction causes a fault in a translation context bank, and that translation context has been configured to assert a context interrupt that is not implemented.

- A Configuration access fault, where a configuration access is made to a non-existent SMMU configuration register.

- A Permission fault, where the permission attributes for a transaction mean it cannot be processed.

- An external fault, where an external abort is reported during the processing of a transaction.

- An unsupported transaction fault, as a result of certain client transactions.

See *Invalid context checking pseudocode on page 3-116* for more information about the Invalid context faults.

3.8.1 Recording a global fault

To record a configuration access fault, the SMMU updates:

- SMMU_sGFSR.
- SMMU_sGFSYNR0, with the following exceptions:
  - Nested is captured as the value 0.
  - NSSTATE is UNK.
- SMMU_sGFAR.
- SMMU_sGFSYNR1 is UNK.

To record an external fault, the SMMU updates SMMU_sGFSR. SMMU_sGFAR, SMMU_sGFSYNR0 and SMMU_sGFSYNR1 are UNK

To record an unsupported upstream transaction fault, the SMMU:

- Updates SMMU_sGFSR, regardless of the state of SMMU_SCR1.GEFRO.
- Aborts the transaction if the SMMU_sCR0.GCFGFRE bit is set to 1.

In an implementation that supports two Security states, SMMU_sGFSR is Banked for security. The Security state of the transaction causing the global fault determines the banked SMMU_sGFSR register that has information about the transaction.

Multiple fault conditions

The Global Fault Status Registers record details of one outstanding fault. If a subsequent fault occurs before an outstanding fault is serviced, the subsequent fault is recorded, but without the full details that accompany the initial fault. SMMU_sGFSR.MULTI indicates the existence of multiple outstanding faults recorded in SMMU_sGFSR.

If a global fault is encountered when SMMU_sGFSR is 0, including SMMU_sGFSR.MULTI, the fault status and syndrome details are recorded in:

- SMMU_sGFSR.
- SMMU_sGFAR.
- SMMU_sGFSYNR0.
- SMMU_sGFSYNR1.

If a global fault occurs when SMMU_sGFSR is nonzero, including SMMU_sGFSR.MULTI, the SMMU_sGFSR.MULTI bit is set to 1 and no other fault status or syndrome state is updated.

Two-stage faults

SMMU_sGFSYNR0.Nested provides information for the following types of fault in SMMU_sGFSR:

- Invalid context fault.
- Unimplemented context bank fault.
- Unimplemented context interrupt fault.

The Nested bit indicates whether the fault condition occurred as a result of either:

- The initial translation function specified by the stream mapping process in SMMU_S2CRn.
- The stage 1 followed by stage 2 translation context specified in the SMMU_CBARn register of the initial context.

Permission faults

Secure instruction fetch transactions can be subject to a protection check. If the SMMU_SCR1.SIF bit is set to 1, a permission fault is recorded when a Secure domain access attempts to exit the SMMU as a Non-secure instruction.

This check applies globally, and if not associated with a translation context bank records faults to SMMU_SGFSR. Otherwise, faults are recorded in the associated SMMU_CBn_FSR.

3.8.2 Handling a Global fault

A transaction is terminated by the SMMU on encountering a global fault. The accesses behave as RAZ/WI. The appropriate Fault Status Register is updated.

3.8.3 Reporting a Global fault

An SMMU can be configured to report a global fault by:

- Reporting client transactions if SMMU_sCR0.GFRE is enabled.
- Reporting configuration transactions if SMMU_sCR0.GCFGFRE is enabled.

Global fault interrupts

An SMMU can be configured to use the following interrupt mechanisms in response to encountering a global fault:

- SMMU_NSgCfgIrpt, to indicate that a configuration access fault has been recorded, where SMMU_sCR0.GCFGFIE enables reporting of the fault. SMMU_sGFSR.CAF specifies whether a configuration access fault has been recorded.
- SMMU_NSgIrpt, to indicate whether any other global fault has been recorded, where SMMU_sCR0.GFIE enables reporting of the fault. SMMU_sGFSR specifies whether such a fault has been recorded.

Note

If SMMU_sGFSR.MULTI is 1 because of a configuration access fault, SMMU_NSgIrpt is asserted instead of SMMU_NSgCfgIrpt.

In an implementation that supports two Security states, the interrupt for communicating these fault classes depends on the Banked copy of SMMU_sGFSR that is updated. If the global fault updates SMMU_sGFSR, interrupt signaling is as previously stated. If the global fault updates SMMU_sGFSR:

- SMMU_gCfgIrpt indicates configuration access faults. SMMU_SCR0.GCFGFIE enables its assertion.
- SMMU_gIrpt indicates all other global faults. SMMU_SCR0.GFIE enables its assertion.

See *SMMU_sCR0, Configuration Register 0 on page 9-189* for more information.

Global faults and untranslated address operations

Client transaction classes that are not subject to translation might be processed by the SMMU. An untranslated address operation might be a barrier transaction, or a TLB maintenance operation that does not specify an address.

When an SMMU processes such a transaction, the transaction remains subject to the stream mapping process. Therefore, any of the following types of fault can occur:

- Invalid context fault.
- Unidentified stream fault.
- Stream match conflict fault.
- Unimplemented context bank fault.

Inclusion of fault reporting in the initiating device and the interconnect between the device and the SMMU is IMPLEMENTATION DEFINED.

3.8.4 Invalid context checking pseudocode

The following pseudocode describes the required order for context bank configuration checks that the SMMU performs after the stream mapping stage. The pseudocode applies to both client transactions and address translation operations.

```
//
// This checks whether a context bank is configured correctly
// If so, it returns true
// Otherwise, it attempts to record a fault in the appropriate FSR and
```

```
// returns false
//

bool CheckBankValidityAndRecordFaults()
{
    if (First is not implemented or is of a different security domain)
    {
        AttemptToRecordGlobalFault( .sGFSR_value(UCBF), .Nested_flag(false) );
        return false;
    }
    else if (
        (First.SMMU_CBAR.TYPE == st1_st2nested && ! SMMU_IDR0.NTS)
        || (First.SMMU_CBAR.TYPE == st1_st2bypass && ! SMMU_IDR0.S1TS)
        || (First.SMMU_CBAR.TYPE == st2_only && ! SMMU_IDR0.S2TS)
        || (First is HW stage 2 only && First.SMMU_CBAR.TYPE != st2_only)
        )
    {
        // Unsupported translation format for the implementation.
        // Chosen to be ICF for SMMUv2
        //
        AttemptToRecordGlobalFault( .sGFSR_value(ICF), .Nested_flag(true) );
        return false;
    }
    else if (
        (First.SMMU_CBAR.TYPE == st1_st2fault)
        || (First is secure && First.SMMU_CBAR.TYPE != st1_st2bypass) // SMMUv1 & SMMUv2
        )
    {
        // S1+S2 nested illegal for secure banks, if a secure bank is
        // not S1+S2 bypass then ICF

        AttemptToRecordGlobalFault( .sGFSR_value(ICF), .Nested_flag(true) );
        return false;
    }
    else if (First.SMMU_CBAR.TYPE == st1_st2nested,
             but Second is not implemented or is of a different security domain)
    {
        AttemptToRecordGlobalFault( .sGFSR_value(UCBF), .Nested_flag(true) );
        return false;
    }
    else if (
        First.SMMU_CBAR.TYPE == st1_st2nested
        && (Second.SMMU_CBAR.TYPE != st2_only
            || (First.SMMU_CBA2R.VA64 && ! Second.SMMU_CBA2R.VA64))
        )
    {
        // S1 Aarch64 and S2 Aarch32 nested combinations are not valid.
        AttemptToRecordGlobalFault( .sGFSR_value(ICF), .Nested_flag(true) );
        return false;
    }
    else if (
        // All ATS operations that are not representable as an upstream bus
        // transaction will TranslationFault.
        //
        // The intent is that the HW need not have a page walk mechanism
        // that can deal with any VA solely for the purposes of supplying
        // ATS functionality.
        Is an ATS operation and not representable_as_upstream_bus(VA)
        )
    {
        // Generate Translation Fault into first context.
        AttemptToRecordContextFault(
            .FSR_value(TF),
            .PLVL_value( SMMU_CBA2R.VA64 ? 0 : 1),
            ...
        );
        return false;
    }
```

```
    else if (
        // Check that VA is <= 32 bits for Stage 1 V7 contexts
        operation has an address        // Not true for barriers
        && First.SMMU_CBA2R.VA64 == 0
        && First.SMMU_CBAR.TYPE in {st1_st2bypass, st1_st2nested}
        && VA > 0xffff_ffff
        )
    {
        // Generate Translation Fault into first context.
        AttemptToRecordContextFault(
            .FSR_value(TF),
            .PLVL_value(1),
            ...
        );
        return false;
    }
    else if (
         // Check that IPA is <= 40 bits for Stage 2 V7 contexts
         operation has an address // Not true for barriers
         && First.SMMU_CBA2R.VA64 == 0
         && First.SMMU_CBAR.TYPE == st2_only
         && IPA > 0xff_ffff_ffff
         )
    {
        // If the MMU is on then this fails the input address check
        // and generates a TranslationFault.
        //
        // If the MMU is off then this fails the output address check
        // and generates an AddressSizeFault
        //
        // NOTE that this is conditional on SMMU_CBn_SCTLR whereas
        // for a VA > 0xFFFF_ffff to a v7s/l stage 1 bank then it
        // unconditionally generates TranslationFault.  This is for
        // backwards compatible behaviour with SMMUv1.
        //
        // This equivalent scenario could not exist in SMMUv1 as the
        // upstream bus size was <= 40 bits and could not occur.
        //
        AttemptToRecordContextFault(
            .FSR_value(First.SMMU_CBn_SCTLR.M ? TF : ASF),
            .PLVL_value(1),
            ...
            );
        return false;
    }

    return true;
}
```

3.9 Configuration access

SMMU configuration is performed through a register space in the system address map. This address map might be partially populated, with the possibility of fully implemented, partially implemented and unimplemented registers throughout that address space.

The following behaviors are permitted for access to an unimplemented register or an unimplemented register bit in the SMMU configuration space:

- Access to an unimplemented configuration register can be RAZ/WI, or can result in a configuration access fault. Such behavior is also permitted for a Non-secure access to a configuration resource that is allocated for use by Secure software.
- Access to an unimplemented bit in a configuration register where at least one bit is implemented can behave as RAZ/WI.

An SMMU implementation can restrict access to some or all of the configuration addresses based on the privilege of the access, with the following behaviors permitted where the transaction does not have sufficient access permissions:

- Treat the access as RAZ/WI.
- Raise a configuration access fault.

Unless explicitly specified in relation to a particular configuration address or range of addresses, it is IMPLEMENTATION DEFINED as to which of the permitted behaviors occurs.

In SMMUv2, accesses to memory locations in a translation context bank cannot result in a configuration access fault. This means that accesses to UNDEFINED translation context bank registers are RAZ/WI, including accesses to:

- Unallocated addresses.
- Any register field that is reserved.

Warning

In SMMUv1, accesses to UNDEFINED translation context bank registers might generate a configuration access fault that corrupts SMMU_sGFSR.

If an implementation generates a configuration access fault in response to Non-secure software accessing a Secure register such as SMMU_SCR1, or a Secure resource such as a context bank allocated for use by Secure software, the fault is reported to SMMU_SGFSR. The fault also:

- Raises an interrupt, if the SMMU_SCR0.GCFGFIE bit is set to 1
- Returns an abort, if the SMMU_SCR0.GCFGFRE bit is set to 1.

In SMMUv2, the SMMU_SCR1.NSCAFRO bit provides an override whereby such configuration access faults are instead recorded in SMMU_GFSR.

For configuration access faults recorded to SMMU_SGFSR, Secure software can read the SMMU_SGFSYNR0.NSSTATE bit to determine whether the fault was generated by an SSD Non-secure transaction. Configuration access faults recorded to SMMU_GFSR do not affect SMMU_SGFSYNR0.

When the SMMU_SCR1.GASRAE bit is set to 1, the global SMMU address space is accessible only by Secure configuration memory accesses. This means that all configuration access faults generated to global SMMU address space registers are also reported to SMMU_SGFSR, unless the SMMUv2 SMMU_SCR1.NSCAFRO bit is set to 1. This permits Secure software to detect potential security violations by Non-secure software.

Note

The SMMU_SCR1.NSCAFRO override prevents Non-secure software from injecting faults into SMMU_SGFSR and preventing Secure software from detecting or managing other faults.

3.10 External faults

A fault encountered outside the SMMU can be reported to the SMMU across the system interconnect.

An external fault encountered in an instruction read, a data read or a data write can be reported to the SMMU, and the SMMU might record the fault synchronously or asynchronously.

External faults in an SMMU implementation are generated in response to:

- A read issued as part of a translation table walk.
- Optionally:
  - A translated client transaction where the downstream system returns an abort.
  - In SMMUv1 only, an unsupported upstream transaction.

  SMMUv2 supports a dedicated unsupported upstream transaction fault. See *Unsupported upstream transaction faults on page 3-103*.

For external faults returned in response to a translation table walk read:

- If the translation table walk is part of processing a client transaction, the SMMU records the external fault in the appropriate Fault Status Register.
- If the translation table walk is part of processing an address translation operation initiated by a configuration access, the fault status is generally captured in SMMU_sGPAR or SMMU_CBn_PAR. An exception to this is where an external fault is encountered during a stage 2 translation table walk that is performed as part of processing an address translation operation initiated in a stage 1 translation context bank. In this case, stage 2 SMMU_CB*n*_FSR and related registers might capture a fault status. Depending on how the fault is serviced, the corresponding stage 1 translation context bank SMMU_CBn_PAR captures an *Address Translation Operation Terminated* (ATOT) status.

In SMMUv2, for external fault that is generated when the downstream system returns an abort, where an implementation supports two Security states and the SMMU_SCR1.GEFRO bit is set to 1, the SMMU records the fault in SMMU_SGFSR. In all other cases, the SMMU records the fault in either

- The appropriate SMMU_sGFSR, if the Security state is known.
- SMMU_GFSR.

Note

Where SMMU_SCR1.GEFRO bit is set to 1, the software agent controlling the Secure space permits Non-secure agents to report faults to the Secure state, potentially interfering with the use of SMMU_SGFSR by the Secure state.

In SMMUv2, an SMMU implementation might record an external fault in response to a translated client transaction if the upstream bus system does not support transaction error reporting. In such cases, synchronous external faults that are associated with a context bank must be recorded in the context bank. All other external faults must be recorded globally.

Note

- In AMBA systems, instead of recording such faults, the SMMU must transfer the faults to the upstream client device. This means that all of the following bits must be RAZ/WI:
  - SMMU_sGFSR.EF.
  - SMMU_SCR1.GEFRO.
  - SMMU_SGFSYNR0.NSSTATE.
- In non-AMBA systems that do not support transferring faults to upstream client devices, Secure software permits Non-secure software to report faults to the Secure state. This can interfere with Secure access to SMMU_SGFSR.

In SMMUv1, for external faults reported to the SMMU in all cases other than those returned in response to a translation table walk read, it is IMPLEMENTATION DEFINED which external faults, if any, the SMMU records.

An external abort reported by the SMMU selects a Fault Status Register group and updates the Fault Status Register in this group with a fault status code. A synchronously recorded external abort selects the Fault Status Register group corresponding to the SMMU resources that processed the faulty transaction. If the transaction bypassed translation, the global fault status group is used. If the transaction was processed with one or more translation context banks, the fault status group belonging to the translation context bank is used.

For stage 1 followed by stage 2 translation, the stage 2 translation context bank is always used.

For an asynchronously recorded external abort, the Fault Status Register group selected is IMPLEMENTATION DEFINED. The selected register group can be a Fault Status Register group as specified for synchronous reporting or the global Fault Status Register group.

A synchronously recorded external abort updates the Fault Status Register and the fault address and Fault Syndrome Registers. The Fault Status Register is updated to indicate that an external fault has occurred. The fault address and Fault Syndrome Registers are updated with syndrome information about the transaction that caused the fault.

An asynchronously recorded external abort updates the Fault Status Register to indicate that an external fault has occurred. The fault address and syndrome registers are UNKNOWN.

In an implementation that supports two Security states, an external fault recorded in the global fault status group can update either SMMU_GFSR or SMMU_SGFSR, depending on the Security state of the transaction that caused the external fault. See *SMMU_sGFSR, Global Fault Status Register on page 9-209*.

For an asynchronously recorded external abort in the global fault status group, all external faults are permitted to be recorded in SMMU_GFSR. Secure software is provided with an override, SMMU_SCR1.GEFRO, which when set, causes all such faults to instead be recorded in SMMU_SGFSR.

3.11 Reporting exclusive access transactions

Depending on the bus system used, an implementation might support exclusive access transactions, in which case the possible return responses indicate that the transaction has:

- Succeeded.
- Failed.
- Aborted.

In the SMMU, reporting means specifically reporting an exclusive access transaction as aborting. When the SMMU is configured so that aborts are not reported, that is, when either the SMMU_sCR0.GFRE bit or the SMMU_CBn_SCTLR.CFRE bit is set to 0, ARM recommends that an implementation reports that the transaction has failed.

3.12 Fault behavior in virtualized context banks

When a hypervisor virtualizes a Guest OS that expects a particular set of virtual upstream client devices, these devices might not map to the set of physical upstream client devices present in the system.

This means that the following Stage 1 context virtualization holes exist in SMMUv1 and SMMUv2:

- The SMMU_CBn_SCTLR.CFRE bit controls whether Context faults encountered when processing an upstream client transaction are reported to the physical device managed by the hypervisor, rather than being reported to the virtual device.
- The SMMU_CBn_SCTLR.CFCFG bit controls whether a fault encountered when processing an upstream client transaction causes the physical device managed by the hypervisor to stall or terminate, rather than the virtual device. This is a problem if the physical device requires use of the stall model and cannot be restarted correctly following a terminated transaction.
- It is possible for non-faulting stage 1 context banks to terminate transactions silently when both of the following apply:
  - Multiple stage 1 context banks use a stage 2 context bank that is configured to use the terminate model.
  - The appropriate stage 1 SMMU_CBn_SCTLR.HUPCF==0.

  This affects other stage 1 banks that use the same stage 2 context. ARM recommends that SMMU_CBn_SCTLR.HUPCF is set to 1 when multiple stage 1 devices share a stage 2 context bank.

  Note

  This also applies where different independent devices use the stage 2 context bank directly.
- Whether broadcast TLB maintenance operations affect SMMU operation depends on the value of the PTM and VMIDPNE bits in SMMU_sCR0. SMMU TLBs are not required to respond to broadcast TLB operations when either:
  - SMMU_sCR0.PTM==1.
  - SMMU_sCR0.VMIDPNE==1.

Chapter 4
Address Translation Operations

This chapter describes address translation operations that are initiated using software-accessible registers. It contains the following sections:

4.1 About address translation operations

Software can access registers to initiate address translation operations. Using a specified translation context bank, software can access address translation registers to initiate:

- Stage 1 translation from a virtual address to an intermediate physical address.
- Single-step translation from a virtual address to a physical address.

The address translation registers are in the global address space and the stage 1 translation context bank address space. They are equivalent to the registers in the processor architecture, with some minor differences in the usage model and in the fault handling.

The remainder of this chapter describes the address translation registers.

4.1.1 Address translation registers in SMMUv2

In SMMUv2, the address translation registers are OPTIONAL. The address translation registers are implemented only when both:

- The SMMU_IDR0.S1TS bit is set to 1.
- The SMMU_IDR0.ATOSNS bit is set to 0.

When not implemented, the address translation operations are reserved locations in the register space. This means that any context bank address translation registers are RAZ/WI, and any global address translation registers are either RAZ/WI or generate a configuration access fault, depending on the implementation.

The input address provided to address translation operations is always independent of the selected translation granule size. That is, the input address must include all high-order bits down to, and including, bit[12]. These bits are translated and provided to the appropriate SMMU_CBn_PAR.

In SMMUv2, when address translation registers are implemented, operations that perform two stages of translation, such as SMMU_sGATS12*, are available even when an implementation does not support stage 1 followed by stage 2 translations.

The address provided to an address translation operation is not subject to sign-extension and software must provide a complete address. See *Sign-extension of input addresses on page 1-38* for more information.

4.2 Address translation registers in a stage 1 translation context

Table 4-1 shows the registers that are available in a stage 1 translation context bank.

Table 4-1 Stage 1 address translation registers

Register	Operation
SMMU_CBn_ATS1UR	Stage 1 unprivileged read
SMMU_CBn_ATS1UW	Stage 1 unprivileged write
SMMU_CBn_ATS1PR	Stage 1 privileged read
SMMU_CBn_ATS1PW	Stage 1 privileged write

To begin translation, software writes the address that is to be translated, to the required register address. The stage 1 translation tables give the result of a successful translation. This result is an intermediate physical address.

4.2.1 Usage model

Address translation operations are initiated using the following registers, from the same translation context bank:

- SMMU_CBn_PAR.
- SMMU_CBn_ATSR.

If the operation succeeds, SMMU_CBn_PAR holds the translated address.

SMMU_CBn_ATSR tracks the progress of an address translation operation. SMMU_CBn_ATSR.ACTIVE is set to 1 when a new address translation operation starts, and remains set at this value until the previously requested operation completes. UNPREDICTABLE behavior arises if an address translation operation is requested in the same translation context bank before the previously requested address translation operation completes.

4.2.2 Fault handling

If an address translation operation fails, SMMU_CBn_PAR generally captures a fault code. See also *Fault handling within stage 1 followed by stage 2 translations initiated in a context bank*.

Note

Address translation operations in a stage 1 translation context operate independently from the translation of client transactions within that stage 1 translation context bank. A fault in a stage 1 context bank, indicated by SMMU_CBn_FSR having a nonzero value, does not restrict address translation operations.

4.2.3 Stage 1 followed by stage 2 translation effect

In an implementation that supports stage 1 followed by stage 2 translation, if a stage 1 translation context bank is associated with a stage 2 translation context bank, the stage 1 address translation operation locates the stage 1 translation tables necessary to perform the operation using the translations specified by the stage 2 translation context bank.

The result of a successful operation is the address specified by the stage 1 translation context bank. This is the intermediate physical address, specific to stage 1 followed by stage 2 translation.

4.2.4 Fault handling within stage 1 followed by stage 2 translations initiated in a context bank

This section applies to address translation operations initiated in a context bank. See *Fault handling on page 4-131* for information about fault handling for global address translations.

If an address translation operation is initiated in a stage 1 translation context bank that is associated with a stage 2 translation context bank, the handling of a fault that is encountered when processing an address translation operation is modified for the following conditions:

- An external abort.
- A fault during stage 2 translation required by a stage 1 initiated address translation.

Depending on the nature of the fault, it is recorded either in the stage 2 context bank or in SMMU_sGFSR. The address translation operation is either suspended or terminated.

Handling faults that are recorded in a stage 2 context bank

This section applies to address translation operations initiated in a context bank. See *Fault handling on page 4-131* for information about fault handling for global address translations.

A fault encountered during an address translation operation initiated in a stage 1 translation context bank is recorded in a stage 2 context bank if it is a result of any of the following conditions:

- An external abort.
- A fault during stage 2 translation required by a stage 1 initiated address translation, including:
  - A stage 2 permission fault.
  - A stage 2 translation fault.
  - An external abort on a stage 2 translation table walk.

In such cases, the address translation operation is either suspended or terminated.

If the fault is recorded, that is, if the value in the stage 2 SMMU_CBn_FSR register is 0 prior to the fault, then:

- SMMU_CBn_FSR captures the appropriate fault status.
- The stage 2 SMMU_CBn_FAR register captures the input address to the address translation operation

  Note

  This is the virtual address, as would be the case for a stage 1 followed by stage 2 client transaction faulting in stage 2.

- The stage 2 SMMU_CBn_FSYNR0 register captures other syndrome information about the fault, including:
  - The ATOF bit, to indicate that the fault is a result of a stage 1 address translation operation.
  - The S1PTWF bit, to indicate that the fault is related to a stage 1 translation table walk.
  - The S1CBNDX bit, that records the stage 1 context bank.
- The value in SMMU_CBFRSYNRAn is UNKNOWN, because no StreamID or SSD_Index can be associated with the address translation operation.

If the fault is not recorded, that is, if the value in the stage 2 SMMU_CBn_FSR register is not 0 prior to the fault, the stage 2 SMMU_CBn_FSR.MULTI bit is set to 1.

If the fault is recorded and stalling is permitted in stage 2, then the address translation operation is stalled in the same way as a client transaction, and:

- The stage 1 SMMU_CBn_ATSR.ACTIVE register bit is not reset, indicating that the operation is active.
- The stage 1 SMMU_CBn_PAR register value remains UNKNOWN.
- The stage 2 SMMU_CBn_RESUME register can be used to resume the operation in stage 2

If the stage 2 context is configured to use the Terminate fault model, the SMMU terminates the address translation operation, as follows:

- The stage 1 SMMU_CBn_ATSR.ACTIVE field resets.
- The stage 1 SMMU_CBn_PAR register updates with the *Address Translation Operation Terminated* (ATOT) fault code, regardless of the value of the stage 2 SMMU_CBn_SCTLR.CFRE bit.

Note

If stage 2 is already faulting and the stage 2 SMMU_CBn_SCTLR.HUPCF bit is set to 0 so that the bank terminates subsequent transactions, then the address translation operation is terminated regardless of whether the operation might have completed successfully.

Handling faults that are recorded in the Global Fault Status Registers

This category of fault handling applies to faults encountered during a stage 2 translation operation that is required by an address translation operation initiated in a stage 1 context bank. These include:

- Stage 2 invalid context faults.
- Stage 2 unimplemented context faults.
- Stage 2 invalid context interrupt faults.

Such faults are never stalled as there is no stage 2 to configure the stall model. The SMMU attempts to record these faults in SMMU_sGFSR

If the fault is recorded, that is, if the value in SMMU_sGFSR is 0 prior to the fault:

- SMMU_sGFSR captures the appropriate fault status.
- In SMMU_sGFSYNR0, the following bits are set to 1:
  - Nested.
  - NSSTATE.
  - NSATTR.
  - ATS.
- In SMMU_sGFSYNR0, the value of the following bits is UNKNOWN:
  - WNR.
  - PNU.
  - IND.
- The value of the following registers is UNKNOWN:
  - SMMU_sGFAR.
  - SMMU_sGFSYNR1.
- SMMU_sGFSYNR2 is updated with an IMPLEMENTATION DEFINED value.

Note

These address translation operations are not recoverable, because these faults are caused by the hypervisor failing to configure the SMMU correctly. The UNKNOWN fields cannot be used for diagnostic purposes.

If the fault is not recorded, that is, if the value in SMMU_sGFSR is not 0 prior to the fault:

- The SMMU_sGFSR.MULTI bit is set to 1 and the other bits in SMMU_sGFSR are not updated
- The following registers are not updated:
  - SMMU_sGFAR.
  - SMMU_sGFSYNR0.
  - SMMU_sGFSYNR1.
  - SMMU_sGFSYNR2.
- The address translation operation is terminated, meaning that:
  - The stage 1 SMMU_CBn_ATSR.ACTIVE bit is reset.
  - The stage 1 SMMU_CBn_PAR register is updated with the ATOT fault code.

4.3 Address translation registers in the global address space

Table 4-2 shows software-accessible registers in the global address space.

Table 4-2 Global address translation registers

Register	Operation
SMMU_sGATS1UR	Stage 1 unprivileged read
SMMU_sGATS1UW	Stage 1 unprivileged write
SMMU_sGATS1PR	Stage 1 privileged read
SMMU_sGATS1PW	Stage 1 privileged write
SMMU_sGATS12UR	Stage 1 and 2 unprivileged read
SMMU_sGATS12UW	Stage 1 and 2 unprivileged write
SMMU_sGATS12PR	Stage 1 and 2 privileged read
SMMU_sGATS12PW	Stage 1 and 2 privileged write

To begin translation, software writes to the required register address. All of the registers require an input address and a stage 1 translation context bank index as arguments.

If a register receives as its input argument, an invalid index from the software, an Unimplemented context bank fault is recorded by the SMMU in the global context bank. An example of an invalid index is an index that corresponds to either:

- A non-existent translation context bank.
- A translation context bank configured in the format of a stage 2 translation context bank.

A Secure register, for example SMMU_SGATS1UR, accepts an index of either a Secure or a Non-secure stage 1 translation context bank. A Non-secure register, for example SMMU_GATS1UR, accepts an index of a Non-secure translation context bank.

In an implementation that supports two Security states, equivalent functionality provides Secure configuration accesses in the form of SMMU_SGPAR, SMMU_SGATSR and SMMU_SGATxx operations. See SMMU_sGPAR, SMMU_sGATSR and Table 4-2 for more information.

The Secure global address translation operations operate independently to the Non-secure global address translation operations. A Secure register accepts an index of either a Secure or a Non-secure translation context bank. A Non-secure register accepts an index of a Non-secure translation context bank. Aliases in the global address space give Secure software access to the Non-secure address translation registers.

See the following sections for more information:

- *Usage model*.
- *Fault handling on page 4-131*.
- *Effect of stage 1 followed by stage 2 translations on page 4-131*.

4.3.1 Usage model

Address translation operations are initiated using the following registers, from the same translation context bank:

- SMMU_sGPAR.
- SMMU_sGATSR.

If the operation is successful, the corresponding SMMU_sGPAR register holds the translated address.

SMMU_sGATSR tracks translation progress. When an address translation operation starts, SMMU_sGATSR.ACTIVE is set to 1, and remains set at 1 until the previously requested operation completes.

If software invokes an address translation operation before the previously requested operation completes, the resulting behavior is UNPREDICTABLE.

Secure address translation operations work in combination with SMMU_SGPAR and SMMU_SGATSR. Non-secure address translation operations work in combination with SMMU_GPAR and SMMU_GATSR.

After accessing any of these registers, software must read SMMU_sGATSR.ACTIVE to determine whether:

- SMMU_sGPAR has been updated to show the outcome of the last requested address translation operation.
- A new address translation operation can be initiated.

4.3.2 Fault handling

If a global translation fails, the corresponding SMMU_sGPAR register captures a fault code. The handling of faults arising from global address translation operations is more simple than that of translation context bank address translation operations, because global address translation operations do not require any consideration for the interaction between a hypervisor and an operating system.

Global address translation operations that are initiated in global register space are not subject to stalls. For these operations, the SMMU reports faults to SMMU_sGPAR rather than the stage 1 or stage 2 context resources, regardless of whether:

- The translation is stage 1 followed by stage 2.
- The context banks identified are configured to stall or terminate.
- The context bank is already faulting.

In addition, these operations are not queued behind client transactions and other translation requests in a stalling context bank that is recording a fault.

If a fault is encountered during a local address translation operation, the SMMU_sGPAR records the type of fault and the address translation stage in which it occurred.

4.3.3 Effect of stage 1 followed by stage 2 translations

In an implementation that supports stage 1 followed by stage 2 translation, if a stage 1 translation context bank is associated with a stage 2 translation context bank, the stage 1 address translation operation locates the stage 1 translation tables necessary to perform the operation using the translations specified by the stage 2 translation context bank.

If the stage 1 translation context bank is associated with a stage 2 fault context, the address translation operation fails and an invalid context fault occurs.

Chapter 5
Coherency Issues and Cache Maintenance Operations

This chapter describes the relationship between controlling software and the SMMU, including optional support for coherent translation table walks. It also describes cache maintenance operations. This chapter contains the following sections:

5.1 Updating the state of a context bank

Controlling software is responsible for coordinating the update of SMMU state so that an update does not interfere with concurrent accesses. Before a write, controlling software must ensure that no transaction that might be affected by a change of SMMU state is in progress, otherwise that transaction might use context state that is inconsistent with some registers having been modified to contain new state, while other registers contain old state.

Note

Any update to a single SMMU register is atomic. Processing a transaction using inconsistent state is possible only after an update to multiple registers.

The SMMU is permitted to make translation table walks for enabled contexts at any time, including speculative accesses.

To prevent the use of inconsistent state, before any non-atomic changes are made, the context must be disabled by clearing SMMU_CBn_SCTLR.M. This prevents any new TLB entries being speculatively allocated. The change to SMMU_CBn_SCTLR.M must be synchronized by issuing TLB invalidates that affect all TLB entries that could have been allocated by the context bank. When the invalidates have been synchronized and completed, the context bank can be changed and then re-enabled with the new configuration.

5.2 Translation table walk coherency

The diverse range of system topologies that an SMMU might be deployed in means that it is not always possible to guarantee a coherent translation table walk. Therefore, it is IMPLEMENTATION DEFINED whether an SMMU translation table walk is coherent.

SMMU_IDR0.CTTW indicates whether support for coherent translation table walks is implemented.

For a system that is being designed to support mainstream platform operating systems, ARM strongly recommends the inclusion of coherent translation table walks.

An ARM processor translation table walk must access its own data or unified cache, or the data or unified cache of another agent participating in the coherency protocol, according to the shareability attributes described in the stage 1 SMMU_CBn_TTBRm or stage 2 SMMU_CBn_TTBR0 register. The shareability attributes described in these registers must be consistent with the shareability attributes for the translation tables.

5.3 Broadcast TLB maintenance operations

Some systems support broadcast TLB maintenance operations, where instead of writing to a memory-mapped register, the SMMU can use a message-based mechanism to apply an operation to all components in the system. For example, in AMBA-based systems, *Distributed Virtual Memory* (DVM) messages can provide broadcast TLB functionality.

The diverse range of system topologies that an SMMU might be deployed in means that it is not always possible to guarantee support for broadcast TLB maintenance operations. Therefore, it is IMPLEMENTATION DEFINED whether an SMMU supports broadcast TLB maintenance operations.

SMMU_IDR0.BTM indicates support for broadcast TLB maintenance operations.

For a system that is being designed to include mainstream platform operating systems, ARM strongly recommends the provision of support for broadcast TLB maintenance operations.

5.3.1 Private VMID namespace

In a system where the system software has a different VMID namespace between the processor and the SMMU, software can hint to the SMMU that any broadcast TLB maintenance operation specifying a VMID is to be ignored. SMMU_sCR0.VMIDPNE provides this hint.

For example, a broadcast Non-secure SMMU_CBn_TLBIVA operation that specifies a VMID is not required to affect the TLB entries. However, a broadcast SMMU_TLBIALLNSNH must affect all TLB entries that are tagged Non-secure and non-hypervisor.

Note

This hint has no effect on Secure broadcast TLB Invalidate operations because they have no associated VMID.

5.3.2 Private ASID namespace

In a system where the system software has a different ASID namespace between the processor and the SMMU, software can hint to the SMMU that any broadcast TLB maintenance operation specifying an ASID is to be ignored. SMMU_CBn_SCTLR.ASIDPNE provides this hint.

5.4 TLB maintenance operations

An SMMU includes memory-mapped registers that can be used to perform TLB maintenance, invalidating translation table entries held in a TLB by the SMMU implementation. The registers for TLB maintenance operations are provided in both:

The translation context bank address space

These registers are for the maintenance of TLB entries associated with a particular translation context.

Note

A Guest OS can use these registers to directly maintain its own translations.

A TLB maintenance operation for these TLB entries is initiated by a write to one of the following registers:

For the maintenance of entries for stage 1 translations:

- *SMMU_CBn_TLBIALL, TLB Invalidate All on page 16-321*.
- *SMMU_CBn_TLBIASID, TLB Invalidate by ASID on page 16-321*.
- *SMMU_CBn_TLBIVA, TLB Invalidate by VA on page 16-322*.
- *SMMU_CBn_TLBIVAA, TLB Invalidate by VA All ASID on page 16-324*.
- *SMMU_CBn_TLBIVAAL, TLB Invalidate by VA, All ASID, Last level on page 16-325*.
- *SMMU_CBn_TLBIVAL, TLB Invalidate by VA, Last level on page 16-326*.

A write to SMMU_CBn_TLBSYNC initiates a TLB maintenance synchronization request, and SMMU_CBn_TLBSTATUS indicates the completion of that request.

For the maintenance of entries for stage 2 translations:

- *SMMU_CBn_TLBIIPAS2, Invalidate TLB by IPA on page 17-355*.
- *SMMU_CBn_TLBIIPAS2L, Invalidate TLB by IPA, Last level on page 17-356*.

A write to SMMU_CBn_TLBSYNC initiates a TLB maintenance synchronization request, and SMMU_CBn_TLBSTATUS indicates the completion of that request.

The global address space

These registers are for the maintenance of TLB entries across the entire SMMU implementation.

Note

Supervisory software can use these registers for TLB maintenance, and can use SMMU_sTLBGSYNC and SMMU_sTLBGSYNC to confirm completion of TLB maintenance operations globally across the SMMU.

A TLB maintenance operation across the entire SMMU is initiated by a write to one of the following registers:

- *SMMU_STLBIALL, TLB Invalidate All on page 9-232*.
- *SMMU_STLBIALLM, Secure TLB Invalidate All MONC on page 9-233*.
- *SMMU_TLBIALLH, TLB Invalidate All Hyp on page 9-233*.
- *SMMU_TLBIALLNSNH, TLB Invalidate All Non-secure Non-Hyp on page 9-233*.
- *SMMU_STLBIVALM, Secure TLB Invalidate MONC by VA, Last level on page 9-233*.
- *SMMU_STLBIVAM, Secure TLB Invalidate MONC by VA on page 9-234*.
- *SMMU_TLBIVAH, Invalidate Hyp TLB by VA on page 9-235*.
- *SMMU_TLBIVAH64, Invalidate Hyp TLB by VA, AArch64 on page 9-235*.
- *SMMU_TLBIVALH64, Invalidate Hyp TLB by VA, Last level, AArch64 on page 9-236*.
- *SMMU_TLBIVMID, TLB Invalidate by VMID on page 9-236*.
- *SMMU_TLBIVMIDS1, TLB Invalidate Stage 1 by VMID on page 9-237*.

A write to SMMU_sTLBGSYNC initiates a TLB maintenance synchronization request. As a minimum, the synchronization operation applies to the specified Security state, and includes all TLB Invalidate operations initiated in context banks associated with that Security state. SMMU_sTLBGSTATUS indicates completion of all the TLB invalidate operations initiated before the most recent write to SMMU_sTLBGSYNC.

5.4.1 Scope of the TLB maintenance operations

The TLB maintenance instructions provide a mechanism for invalidating entries from TLB caching structures to ensure that changes to the translation tables are reflected correctly in the TLB caching structures.

The architecture permits the caching of any translation table entry that has been returned from memory without a fault and that does not, as a result of that entry, cause a Translation Fault or an Access Flag fault. This includes:

- Entries in translation tables that point to subsequent table to be used in that stage of translation.
- Stage 2 translation table entries used as part of a stage 1 translation table walk.
- Stage 2 translation table entries used to translate the output address of a stage 1 translation.

Such entries might be held in intermediate TLB caching structures that are used during a translation table walk that are distinct from the data caches in that they are not required to be invalidated as the result of writes of the data.

The architecture does not impose any restriction on the form of any TLB caching structures used for holding translation table entries. In particular for translation regimes that involve two stages of translation, it recognizes that such caching structures at any level of the translation table walk might contain:

- Entries containing information from stage 1 translation table entries.
- Entries containing information from stage 2 translation table entries.
- Entries combining information from both stage 1 and stage 2 translation table entries.

The TLB maintenance operations associated with the translation context bank address space for stage 1 translations apply to any cached entries that include any stage 1 information that would be used to translate the address or context being invalidated.

Note

- Where stage 1 information has been cached in multiple entries, as could occur from splintering of a page when caching in the TLB, the invalidation must apply to all cached entries containing stage 1 information from the page that is used to translate the address being invalidated, regardless of whether that cached entry would be used to translate the address being invalidated.
- ARM expects that, in at least some implementations, cached copies of levels of the translation table walk other than the final level will be tagged with their ASID, regardless of whether the final level is global or non-global, and so TLB invalidations that involve the ASID will require the ASID to match the entry for that entry to be invalidated.

The TLB maintenance operations associated with the global address space apply to any cached entries in the caching structures that include any stage 1 translation table entries and would be used when translating the address or context being invalidated. *The global address space on page 5-137* list these operations.

The following TLB maintenance operations associated with the global address space also apply to any cached entries in caching structures that both include any stage 2 translation table entries and would be used when translating the address or context being invalidated:

- *SMMU_TLBIALLNSNH, TLB Invalidate All Non-secure Non-Hyp on page 9-233*.
- *SMMU_TLBIVMID, TLB Invalidate by VMID on page 9-236*.

Whenever translation table entries associated with a particular VMID or ASID are changed, the corresponding entries must be invalidated from the TLB to ensure that these changes are visible to subsequent execution, including speculative, execution that uses those translation table entries.

5.4.2 TLB maintenance operation processing

TLB maintenance operations initiated by a register access have a post and synchronize model, where a TLB Invalidate operation is requested by writing to the appropriate register, and completion is confirmed by a subsequent synchronization operation.

An accepted TLB Invalidate operation is not complete until:

- Every translation held in a TLB that is a target of a TLB Invalidate operation is discarded.
- Every transaction already in progress that has used the translations held in the TLB is globally observed.

A stage 1 descriptor fetch performed as part of a stage 1 followed by stage 2 translation (from initial translation of an IPA to the return of the descriptor data) is considered a single transaction. This means:

- A stage 2 only invalidation may complete when the stage 1 descriptor fetch has completed.
- If the final stage 1 translated transaction uses the affected stage 2 entries for its IPA to PA translation, then the stage 2 TLB invalidation operation does not complete until the stage 1 translation has completed.

An SMMU must accept an unbounded number of memory-mapped TLB maintenance operations without relying on the forward progress of client transactions.

Software can use a SYNC operation to determine that a memory-mapped TLB maintenance operation is complete. After issuing the SYNC operation, software polls the TLB status until the context bank is no longer active.

Note

Software must ensure that it limits the number of TLB Invalidate and SYNC operations issued to the same TLB invalidation resource. Failure to adhere to this can result in a situation where new operations are continuously added at a rate that prevents all operations being completed, preventing the TLB status from reporting that the context bank is inactive.

A SYNC operation accepted in a translation context bank only ensures the completion of a TLB maintenance operation accepted in that translation context bank. A SYNC operation accepted in the global address space ensures the completion of a TLB maintenance operation accepted in either the global address space or in any translation context bank.

POST

The outline assembly language source code for posting a new TLB Invalidate operation is as follows, assuming the operation is to invalidate by virtual address:

```
MOV     R0,#VA
MOV     R1,#SMMU_CBn_TLBIVA
STR     R0,[R1]
```

SYNC

The outline assembly language source code for ensuring the completion of one or more posted TLB Invalidate operations is as follows:

```
    MOV     R0,#SMMU_CBn_TLBSYNC
    MOV     R1,#SMMU_CBn_TLBSTATUS
    STR     R0,[R0] ; Initiate TLB SYNC
Loop:
    LDR     R0,[R1]
    ANDS    R0,R0, #1 ; TLB SYNC ACTIVE STATUS
    BNE     Loop
    DSB
```

5.4.3 Thread safety

TLB maintenance operations initiated by a register access do not provide atomic behavior. However, the state necessary for initiating and tracking the completion of a TLB maintenance operation is duplicated so that multiple threads can work independently:

- In an implementation that supports two Security states, the global TLB maintenance operation state is banked by security.
- The translation context bank TLB maintenance operation state is provided per translation context bank.

Software arbitration is required if there is the potential for multiple threads of activity to use the same TLB maintenance operation state concurrently. For example, a software lock is required if it is possible for multiple threads to attempt to perform TLB maintenance operations in a single translation context bank.

5.5 Cache maintenance operations

In SMMUv2, client devices might perform cache maintenance operations on data and instruction caches, such as evicting data to memory. The stage 1 SMMU_CBn_SCTLR.UCI bit can define whether such operations are permitted from EL0.

Note

Cache maintenance operations are not classed as instruction fetches, and are therefore not subject to SIF permission faults.

The SMMU performs permissions checks on attempts by a client device to initiate a cache maintenance operation. The following pseudocode describes these SMMU permission checks.

```
CheckCacheMaintenanceForPermissionFaultAndRecordFault()
{
    //
    // This function only decodes a Permission Fault for cache maintenance
    // operations (CMOs).
    //
    // The intent is that a cache maintenance operation and a normal access are
    // treated identically except for the permission model.
    //
    // Thus the following code does _not_ include the reporting of:
    //    * External Faults on page walks
    //    * Translation Faults
    //    * Access Faults
    //    * Address Size Faults
    // which would be reported and treated almost identically to normal accesses.
    //
    // The cache maintenance operation permission model may take part in two
    // separate phases -- an early check that might fault even before any
    // descriptor is fetched (and hence, for example, could Permission Fault
    // before an External Fault could occur), and a check that occurs at the
    // same time as if the operation was a data/instruction access.  This is not
    // fully expressed in the pseudo-code below.
    //

    //
    // The SMMU Architecture cache maintenance permission model has to cope with
    // the full outer product of:
    //     - cache invalidate (in ACE MakeInvalid)
    //     - cache clean and invalidate (in ACE CleanInvalid)
    //     - cache clean (in ACE CleanShared)
    //   and
    //     - User
    //     - Privileged
    //   and
    //     - Instruction
    //     - Data
    //   and
    //     - Point of Unification
    //     - Point of Coherency
    //
    // Much of the CMO permission model is implementation defined (IMP DEF) to
    // accommodate the variety of bus standards, types of caches, devices and
    // topological needs of the SoC.  In addition, reads, writes and CMO for a
    // device might be mapped to different contexts by incorporating that
    // information into the stream id.
    //
    // The SMMU architecture provides minimal safety guarantees for CMOs:
    //
    //    * in SMMUv1, then the SMMU can destroy data by using cache-invalidate.
    //      However, if there is a stage 2 and it is turned on then write
    //      permissions must exist for this to be able to occur.  Under the core
```

```
//      architecture then this is reasonable as the guest OS could have used
//      a write to alter the data.  However, in the SMMU architecture, then
//      there is no guarantee that CMOs would have been routed to the same
//      context as writes, so technically one cannot deduce that a write
//      from the device could have altered data.  It is up to the programmer
//      to ensure the required semantics are maintained.
//
//    * in SMMUv2, all cache invalidate operations CMOs will be upgraded to
//      clean-and-invalidate operations.  Thus, under no circumstances will
//      SMMUv2 provide a mechanism for a cache maintenance operation to
//      destroy data.  In SMMUv2, then permission faults on CMOs the user to
//      detect when a device generates CMOs when it is not expected to and
//      potentially catch programming errors.
//
//    * the only allowed IMP DEF behavior is to produce more permission
//      faults or to upgrade a clean-cache maintenance operation to
//      clean-and-invalidate.
//
//    * IMP DEF behavior must not lead to security violation.
//
//    * IMP DEF permission faults may only use the state in:
//        * the transaction
//        * the matching SMMU_S2CRn.INSTCFG/PRIVCFG
//        * SMMU_CBA2Rn.VA64
//        * SMMU_CBA2Rn.MONC
//        * SMMU_CBARn.HYPC
//        * SMMU_CBn_SCTLR.UCI for stage 1 context
//        * SMMU_CBn_SCTLR.M
//        * SMMU_CBn_ACTLR for any involved context banks
//        * SMMU_sACR, with the SMMU_ACR not being able to affect
//          SSD Secure CMOs.
//        * the read, write and execute permissions for non-CMO
//          operations.  This may indirectly depend on
//          SMMU_CBn_SCTLR.UWXN/WXN
//
//    * IMP DEF permission faults must not allow a guest OS to
//      distinguish that it is running under a stage 1 followed
//      by stage 2 context where the MMU is on for stage 2.
//      That is, any permission faults reported to stage 1 should
//      be identical to if the operation was received by a
//      stage 1 with stage 2 bypass context bank.
//      The hypervisor should not be required to fix up a permission
//      fault to inject into the stage 1 for these cases.
//
// NOTE that the architecture does not allow permission faults generated by
// cache maintenance operations to be distinguished from normal
// data/instruction accesses.  An implementation might chose to provide this
// information in SMMU_CBn_FSYNR1.
//

//
// NOTE: SIF checks do not apply to any CMOs as they are not instruction
// "fetches"
//

if (bypassed before context bank determined)
{
    UpgradeToCacheCleanAndInvalidate();
    return allowed;
}

if (SMMUv1)
{
    if (stage 2 exists && stage 2 MMU is on)
    {
        if (no write permissions at stage 2
            && operation is cache invalidate to PoC/PoU)
        {
```

```
                if (imp def option)
                {
                    return st2.PermissionFault();
                }
                else
                {
                    UpgradeToCacheCleanAndInvalidate();
                }
            }
        }

        if (imp def option)
            // There is always the IMP DEF option to upgrade everything to
            // cache clean and invalidate.
            UpgradeToCacheCleanAndInvalidate();

        return allowed;
    }
    else
    {
        // SMMUv2

        if (stage 1 exists and imp def permission fault stage 1)
        {
            return st1.permission_fault();
        }

        if (stage 2 exists and imp def permission fault stage 2)
        {
            return st2.permission_fault();
        }

        // SMMUv2 always upgrades cache-invalidate to cache-clean-and-invalidate
        // and so no data should be able to be destroyed and no permission
        // faults are required.

        if (operation is cache invalidate)
            // cache invalidate is always upgraded
            UpgradeToCacheCleanAndInvalidate();

        if (imp def)
            // it is imp def if all operations are upgraded.
            UpgradeToCacheCleanAndInvalidate();
        return allowed;
    }
}
```

Chapter 6
SMMU Performance Monitors Extension

This chapter describes the SMMU Performance Monitors Extension. It contains the following sections:

6.1 About the SMMU Performance Monitors Extension

The SMMU Performance Monitors Extension is an OPTIONAL memory-mapped extension. If required, the extension can be implemented as a CoreSight component, by including the CoreSight Component and Peripheral ID registers defined in the *CoreSight™ Architecture Specification*. Whether an SMMU includes the Performance Monitors Extension is IMPLEMENTATION DEFINED. If not supported, the register map corresponding to the Performance Monitors registers is UNK/SBZP.

If implemented, the SMMU Performance Monitors Extension provides event counter resources and event filtering based on a translation context or a StreamID. Event counter resources are revealed in the address map of a translation context bank. The configuration of these resources permits a Guest OS to use them without the intervention of a hypervisor.

There are no facilities for Secure software to reserve performance monitoring resources. If required, this can be achieved through a collaborative software protocol between the Secure and the Non-secure software domains.

By default, an event that results from processing a Secure transaction does not contribute to performance monitor counting. SMMU_SCR1.SPMEN enables the counting of such events, as does the external Secure PL1 Non-invasive debug enable input signal, **SPNIDEN**.

6.2 The register map

The SMMU Performance Monitors Extension supports a memory-mapped register interface in the SMMU global address space. This interface occupies a PAGESIZE region starting at an offset of (3 × PAGESIZE) from the SMMU base address. See Chapter 8 *SMMU Address Space* and *PAGESIZE and NUMPAGENDXB on page 8-164* for more information.

The SMMU Performance Monitors Extension also provides the capability to reveal a group of performance monitoring resources in the register map of a translation context bank. See *Translation context bank assignment on page 6-152* for more information.

6.3 Event classes

Table 6-1 shows the event classes of the SMMU Performance Monitors Extension.

Table 6-1 SMMU performance monitoring events

Category	Event number	Description
Cycle	0x0000	Cycle count, occurs every SMMU clock cycle
	0x0001	Cycle count divided by 64 event, occurs every 64th SMMU clock cycle
TLB	0x0008	TLB Entry Allocated, occurs when an SMMU allocates a TLB entry to load a translation
	0x0009	TLB Entry Allocated for a Read or far-atomic
	0x000A	TLB Entry Allocated for a Write
Access	0x0010	Access, occurs when an SMMU processes a new transaction
	0x0011	Access Read
	0x0012	Access Write
IMPLEMENTATION DEFINED	0x0080-0xFFFF	Reserved for IMPLEMENTATION DEFINED events.

It is IMPLEMENTATION DEFINED which event classes are included in an implementation. See *PMCEID0, Performance Monitors Common Event Identifier 0 register on page 12-259* for more information.

All unused encodings are reserved:

- Unused encodings in the range 0x00-to 0x7F are reserved for future architectural event classes.
- Unused encodings in the range 0x80-to 0xFFFF are reserved for IMPLEMENTATION DEFINED event classes.

6.4 StreamID groups

The SMMU Performance Monitors Extension includes the concept of a StreamID group. A StreamID group is a set of StreamIDs. Transactions with a StreamID that is a member of a StreamID group are associated with that group. The number of StreamID groups, and the mapping of StreamIDs to StreamID groups, is IMPLEMENTATION DEFINED.

Event counters are affiliated with a StreamID group. An event counter can only count events caused by the processing of transactions associated with that group.

It is permissible to define a number of StreamID groups with potentially overlapping membership. A global StreamID group can be defined to contain all StreamIDs as members. ARM suggests StreamID group 0 be reserved for this purpose.

The concept of StreamID groups caters for distributed systems where a remote TLB might service only a subset of client devices, therefore only having visibility of a subset of transactions processed by the SMMU. The StreamID group definition makes counting events limited to such a subset of transactions permissible.

Event counters are affiliated with a StreamID group on a fixed basis. There is no mechanism to change the relationship. The affiliation with a StreamID is made on a per-counter group basis.

6.5 Counter groups

The SMMU Performance Monitors Extension provides:

- Translation context event filtering.
- StreamID event filtering.
- Translation context bank assignment.

These mechanisms are configured by controls that operate on a group of counters known as a Counter group.

The number of event counters implemented, and the number of Counter Groups, is IMPLEMENTATION DEFINED:

- PMCFGR.N indicates the total number of implemented event counters.
- PMCFGR.NCG indicates the number of Counter groups.
- For each Counter group, PMCGCRn.CGNC indicates the IMPLEMENTATION DEFINED number of event counters associated with Counter group *n*.

The SMMU architecture permits a maximum of:

- 256 implemented event counters in total.
- 256 Counter groups.
- 15 counters in a Counter group.

Event counters are arranged by their Counter group, with counters associated with Counter group 0 occurring first in the event counter address map.

PMCGCRn and PMCGSMRn configure the following features by Counter group:

- Translation context event filtering.
- StreamID event filtering.
- Translation context bank assignment features.

6.6 Event filtering

A Counter group counts events on a global, translation context, or StreamID basis:

- If configured to count all events, it considers all transactions associated with the StreamID group that the Counter group is affiliated with.
- If configured to filter events on a translation context basis, it only considers transactions processed by the SMMU translation context designated by PMCGCRn.NDX, and limited by the scope of the StreamID group that the Counter group is affiliated with.
- If configured to filter events on a StreamID basis, it only considers transactions that match the ID and the mask designated by PMCGSMRn.ID and PMCGSMRn.MASK, and limited by the scope of the StreamID group that the Counter group is affiliated with. An implementation must provide the same number of PMCGSMRn.ID and PMCGSMRn.MASK bits for every implemented PMCGSMRn register.

PMCGCRn.TCEFCFG selects the type of filtering on which the event counting is based.

6.7 Translation context bank assignment

The SMMU Performance Monitors Extension provides the capability to reveal a Counter group in the register map of a translation context bank. A hypervisor can use this functionality to give a Guest OS direct access to performance monitoring resources, to profile the behavior of any device associated with the translation context that the Guest OS uses.

A Counter group and its related registers can be revealed in a translation context bank register map using the PMCGCRn registers associated with that Counter group:

- PMCGCRn.CBAEN enables translation context bank assignment.
- If enabled by PMCGCRn.CBAEN, PMCGCRn.NDX specifies the translation context bank to which to assign the Counter group.

Only one Counter group can be revealed in a translation context bank. Behavior is UNPREDICTABLE if multiple PMCGCRn registers specify the same translation context bank.

Translation context bank assignment can only be enabled after the following event filtering modes of operation are configured:

- Translation context event filtering.
- StreamID event filtering.

Behavior is UNPREDICTABLE if translation context bank assignment is enabled when global event monitoring and filtering are enabled.

PMCGCRn and PMCGSMRn are not revealed in the translation context bank register map. This would cause a virtualization hole.

When a Counter group is revealed in a translation context bank, one of the registers revealed is an SMMU_CBn_PMCR register associated with that group. This is a Banked register.

6.7.1 Translation context register map

The translation context bank register map occupies an address space defined by PAGESIZE. When a Counter group is revealed in a translation context bank register map, a more compressed register layout balances the use of available space versus future expansion. The layout of performance monitor controls in a translation context bank register map places an upper limit on the number of event counters that can exist in a Counter group.

A Counter group that is revealed in a translation context register map is self-contained. The event counters revealed in the translation context register map are numbered beginning at 0. The Performance Monitor ID and configuration registers in the translation context register map indicate the number of event counters contained in the Counter group, not the total number implemented in the SMMU.

6.7.2 SMMU_CB*n*_PMCR banking

PMCR only operates on event counters of Counter groups that are not revealed in a translation context bank.

A separate register, SMMU_CBn_PMCR, controls event counters in a Counter group that is revealed in a translation context bank. This register is revealed as part of the designated translation context bank. For state save and restore purposes, the active RW bits of this Banked register are available in the PMCGCRn register associated with that Counter group.

6.7.3 Counter group absent

If no Counter group is revealed in a translation context bank, default read-only RAZ/WI values are presented at the Performance Monitors register locations in that translation context bank.

6.8 Event counter overflow interrupt

The SMMU Performance Monitors Extension provides the capability to assert an interrupt in the event of an event counter overflowing. PMINTENSETx and PMINTENCLRx enable and disable interrupt assertion on a per event counter basis.

The interrupt that is asserted depends on the Counter group that the event counter is a member of. One interrupt output is provided per Counter group.

These interrupt outputs are separate from the global and translation context interrupt outputs, because different software modules are generally involved in performance monitoring.

Chapter 7
SMMU Support for Two Security States

This chapter describes the relationship between the SMMU architecture and OPTIONAL support for two Security states. It contains the following sections:

7.1 Sharing resources between Secure and Non-secure domains

When the SMMU is implemented in a system with an ARM processor that supports two Security states, SMMU support for two Security states is OPTIONAL. In an SMMU implementation that supports two Security states, the resources of an SMMU are shared between Secure and Non-secure domains. For information about processor support for two Security states, see the *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

The SMMU architecture permits support for two Security states to be an IMPLEMENTATION DEFINED choice. SMMU_IDR0.SES indicates whether the SMMU supports two Security states.

7.2 Providing SMMU support for only a single Security state

An SMMU implementation might support only a single Security state. However, this does not mean that a transaction from a Secure device cannot arrive at the SMMU. If an SMMU does not support two Security states, and is expected to receive and process any transaction from a Secure device, SSD must ensure that this transaction bypasses all subsequent SMMU transaction processing.

If SMMU_IDR0.SES is 0:

- The SMMU implementation does not translate transactions from Secure devices.
- Stream Match Register groups, translation context banks, and interrupts are not allocated to a Security state.
- The Secure Banked control and status registers are not present.

Note

Regardless of whether an SMMU implementation supports two Security states, the introduction of an SMMU must not create any type of security loophole.

7.3 Providing SMMU support for two Security states

If SMMU_IDR0.SES is 1, Secure software might arrange for an SMMU to process and translate transactions from one or more Secure devices.

7.3.1 Translation restrictions

In an SMMU implementation that supports two Security states, the following restrictions apply to a transaction from a Secure device:

- Transactions must be processed using a stage 1 with stage 2 bypass translation context.
- The stage 1 context must be allocated for use by Secure software.

7.3.2 Resource allocation

In an SMMU implementation that supports two Security states, a number of resources are partitioned between Secure and Non-secure domains. For some of these resources, Secure software might allocate some or all of the resource for the sole use by the Secure software. Such shared resources include:

- Stream mapping register groups. See SMMU_SCR1.NSNUMSMRGO.
- In SMMUv1, context interrupts. See SMMU_SCR1.NSNUMIRPTO.
- Translation context banks. See SMMU_SCR1.NSNUMCBO.

The Non-secure view of the SMMU_IDR*x* registers takes into account the number of registers allocated for use by Secure software when reporting the number of resources. See *SMMU_IDR0-7, Identification registers on page 9-182*.

If Secure software allocates an SMMU resource for use in a particular Security state, one of the following generally applies:

- The reservation occurs at Secure system boot time and is static for the duration of system uptime.
- A software interface between Secure and Non-secure domains is implemented that supports the dynamic partitioning of SMMU resources.

As a consequence of the restrictions specified in *Translation restrictions*, the following conditions apply:

- A transaction that is determined to be SSD Secure is matched only against Stream mapping register groups that are allocated for use by Secure software.
- A Stream mapping register group that is allocated for use by Secure software must specify only one of the following:
  - A translation context bank allocated for use by Secure software.
  - Bypass mode.
  - The fault context.
- In SMMUv1, the SMMU_CBARn register associated with a translation context bank allocated for use by Secure software must only specify a Context interrupt that is allocated for use by Secure software.
- Any translation context bank allocated for use by Secure software must be placed above the translation context bank indicated by SMMU_IDR1.NUMS2CB. This field indicates the last translation context bank that only supports the stage 2 translation format.

In SMMUv2, whether a context bank uses the AArch64 translation regime is controlled by SMMU_CBA2Rn.VA64. When a context bank changes Security state, SMMU_CBA2Rn.VA64 becomes UNKNOWN.

SMMUv2 compliant Secure software must always initialize this bit to an appropriate value:

- After changing a set of context banks from Secure to Non-secure, if Non-secure software executes in AArch32 state, Secure software must set SMMU_CBARn.VA64 to 0. This means that SMMUv1 Non-secure software continues to work without any change.

- After changing a set of context banks to Secure, the Secure software must set SMMU_CBARn.VA64 to the value that corresponds to the Execution state of the Secure software that uses the context banks:
  - If this software executes in AArch32 state, it must set SMMU_CBARn.VA64 to 0.
  - If this software executes in AArch64 state, it must set SMMU_CBARn.VA64 to 1.

SMMUv1 Secure software running on SMMUv2 must be updated so that when it changes the Security state of a set of context banks then it sets SMMU_CBARn.VA64 in this way.

Note

If Secure software does not update SMMU_CBARn.VA64 then a security violation might occur. For example, if Secure software uses a context bank reassigned from Non-secure use, then SMMU_CBARn.VA64 might be set to 1, and so SMMUv1 compliant Secure software assumes that it is using an AArch32 context bank, but it is being interpreted by the SMMU as an AArch64 translation regime.

7.3.3 Permitted transaction resource usage

This section specifies whether a transaction is permitted to interact with Secure or Non-secure resources, or both.

Processor support for two Security states provides a Secure domain and a Non-secure domain, based on the following fundamental concepts:

- The Secure domain must be able to operate in isolation from the Non-secure domain.

  This implies that the Secure domain must have access to registers, instruction memory, and data memory that the Non-secure domain does not have access to. This is achieved by a combination of dedicated Secure resources and shared resources that the Secure domain acts as a gatekeeper for. In the SMMU architecture, this concept is extended by the ability of Secure software to allocate the following SMMU resources to a particular Security state:
  - Stream mapping register groups.
  - Translation context banks.
  - Context interrupts.

- Secure and Non-secure domains must be able to communicate. This is facilitated primarily by permitting the Secure domain to access Non-secure memory.

Under the processor architecture, the basic model is extended in the following ways:

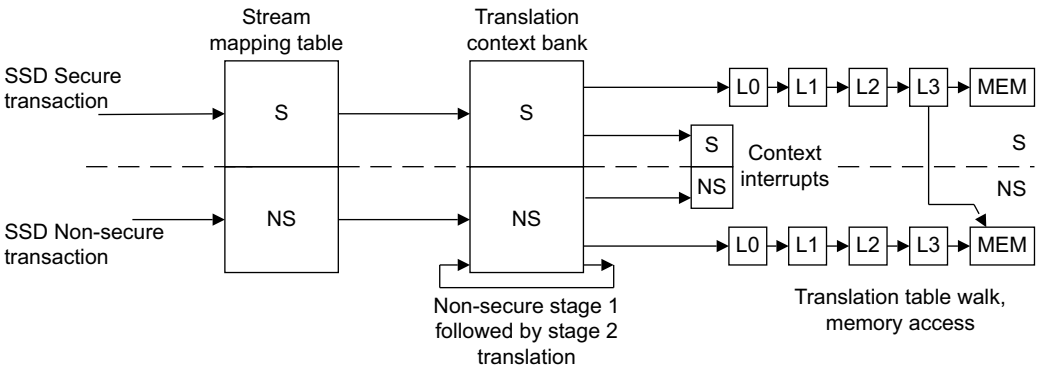
- It is generally useful to give the Secure domain read and write access to any Non-secure domain state in the system.

  Note

  Read and write access to the Non-secure state is not equivalent to being able to use that state in all cases. For example, Secure software can read from and write to the Non-secure SMMU_CBn_TCR registers, but cannot execute an LDR or STR instruction using those registers.

- Inside Non-secure memory, the Secure domain can store translation table mappings that specify translations to Non-secure memory. This reduces the requirements to use Secure memory, which is generally an on-chip, and therefore expensive, resource.

Figure 7-1 on page 7-160 shows the relationship between transaction processing and resource usage. The connections shown are required for enabling basic and fundamental operation.



S = Secure resource managed by Secure software

NS = Non-secure resource managed by Non-secure software

Figure 7-1 Relationship between processing transactions and using resources

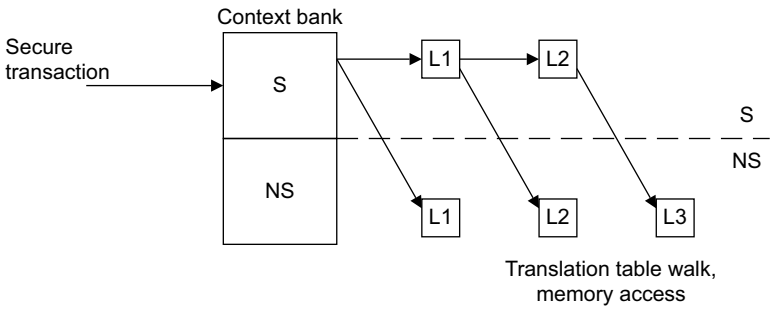
With regard to boundary control between Secure and Non-secure resources:

- Secure software controls the boundary between Secure and Non-secure resources.
- In the Stream mapping table, Secure software can claim Stream mapping register groups using SMMU_SCR0.NSNUMSMRGO.
- In the translation context bank space, SMMU_SCR0.NSNUMCBO enables Secure software to adjust the number of translation context banks visible to Non-secure accesses. See *Resource allocation on page 7-158* for more information and warnings on how to change the number of context banks allocated to each Security state.
- For SMMUv1, in the Context interrupts space, SMMU_SCR1.NSNUMIRPTO can be configured to allocate some of the interrupts for use by Secure software.

In addition to the basic relationship between transaction processing and resource usage, the following connections are permitted:

- Connections that are not ruled out by security requirements.
- Connections that have existing equivalent functionality.
- Connections that are permitted because of other perceived benefits.

Figure 7-2 illustrates these connections.



S, L1, L2 = Secure resource managed by Secure software

NS, L1, L2, L3 = Non-secure resource managed by Non-secure software

Figure 7-2 Permitted relationships between Secure and Non-secure resource usage

Figure 7-1 and Figure 7-2 show all permitted relationships between Secure and Non-secure resource usage. No other relationships are permitted.

Prohibited relationships

Security requirements mean that the following relationships are not permitted:

- A transaction originating from a bus master operating for the Non-secure domain must never be permitted to map to a transaction stream or translation context in the Secure part of the Stream mapping table.
- A transaction associated with the Non-secure part of the Stream mapping table must never be associated with a Secure translation context bank.
- A transaction associated with a Non-secure translation context bank must never be associated with a Secure Context interrupt.
- A transaction associated with a Non-secure translation context bank must never be associated with a Secure level 1 translation table walk
- A transaction in the Non-secure level 1 part of a translation table walk must never be permitted to enter the Secure level 2 part of a translation table walk
- A transaction in the Non-secure level 2 part of a translation table walk must never be permitted to enter the Secure level 3 part of a translation table walk.
- A transaction in the Non-secure level 3 part of a translation table walk must never be associated with target memory managed by Secure software.

In addition, the SMMU architecture prohibits:

- A transaction originating from a bus master operating for the Secure domain must not be permitted to map to a transaction stream or translation context in the Non-secure part of the Stream mapping table.
- A transaction associated with the Secure part of the Stream mapping table must not be associated with a Non-secure translation context bank.
- Stage 1 followed by stage 2 translation in the Secure part of the translation context bank.
- In SMMUv1, a transaction associated with a Secure translation context bank must not be associated with a Non-secure Context interrupt.

Chapter 8
SMMU Address Space

This chapter specifies the address space of an SMMU implementation. It contains the following sections:

8.1 About the SMMU address space

An SMMU is configured through a memory-mapped register frame. The total size of the SMMU address depends on the number of implemented translation contexts.

The SMMU address map consists of the following equally sized portions:

- The global address space.
- The translation context bank address space.

The global address space is at the bottom of the SMMU address space, SMMU_BASE, where SMMU_BASE is aligned to (PAGESIZE * NUMPAGE * 2). The translation context bank address space is located above the top of the global address space, as shown in Figure 8-1.

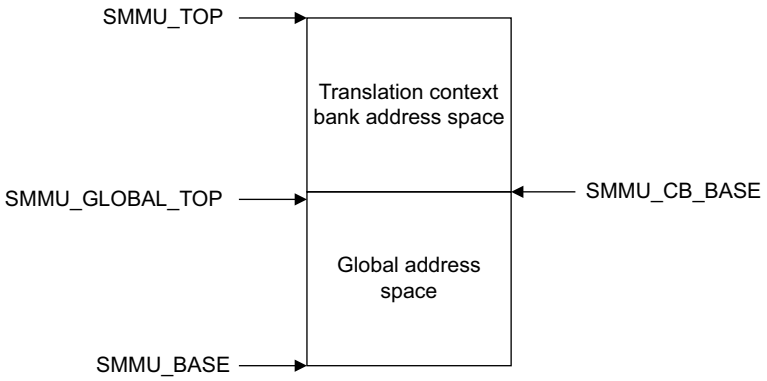


Figure 8-1 SMMU address space

The following register fields indicate the size of the SMMU address space:

- SMMU_IDR1.PAGESIZE.
- SMMU_IDR1.NUMPAGENDXB.

8.1.1 PAGESIZE and NUMPAGENDXB

An SMMU register map arranges state into a number of pages. Each page occupies, and is aligned to, a PAGESIZE space in the address map. Such organization permits a hypervisor to permit or deny access to SMMU state on a page-by-page basis.

The SMMU architecture permits an implementation to support either 4KB or 64KB PAGESIZE options.

SMMU_IDR1.PAGESIZE specifies the implemented PAGESIZE.

NUMPAGE, the number of pages implemented in the global address space or translation context bank address space is determined from SMMU_IDR1.NUMPAGENDXB, where NUMPAGE = $2^{(NUMPAGENDXB+1)}$.

8.1.2 Address space calculation

The following byte address calculations give the sizes and base addresses in the SMMU address space:

- SMMU_GLOBAL_SIZE = SMMU_CB_SIZE = (NUMPAGE × PAGESIZE).
- SMMU_TOP = SMMU_BASE + SMMU_GLOBAL_SIZE + SMMU_CB_SIZE − 1.
- SMMU_GLOBAL_TOP = SMMU_BASE + SMMU_GLOBAL_SIZE − 1.
- SMMU_CB_BASE = SMMU_BASE + SMMU_GLOBAL_SIZE.
- SMMU_CBn_BASE = SMMU_CB_BASE + (n × PAGESIZE).

8.2 The global address space

In an implementation that supports two Security states, the global address space is generally accessible by both Secure and Non-secure configuration accesses. Some address ranges can only be accessed by Secure configuration accesses.

Several registers are Banked. The security status of a configuration access selects the appropriate resource. Alias registers are provided at distinct addresses to give Secure software access to Non-secure versions of the resources.

Setting SMMU_SCR1.GASRAE to 1 enables a restricted access mode of operation. In this mode, with the possible exception of the IMPLEMENTATION DEFINED address space, all of the global address space and the stage 2 format translation context banks are accessible by Secure configuration access only. It is IMPLEMENTATION DEFINED whether SMMU_SCR1.GASRAE only permits Secure accesses to the IMPLEMENTATION DEFINED address space.

These restrictions are in addition to any underlying Secure-only resource that might exist.

Table 8-1 shows the SMMU global address space.

Table 8-1 SMMU Global address space

Offset from SMMU_BASE	Description	Notes
0x00000 to ((1 × PAGESIZE) − 0x4)	SMMU Global Register Space 0	PAGESIZE global address space
(1 × PAGESIZE) to ((2 × PAGESIZE) − 0x4)	SMMU Global Register Space 1	PAGESIZE global address space
(2 × PAGESIZE) to ((3 × PAGESIZE) − 0x4)	SMMU IMPLEMENTATION DEFINED address space	PAGESIZE global address space
(3 × PAGESIZE) to ((4 × PAGESIZE) − 0x4)	SMMU performance monitoring address space	PAGESIZE global address space
(4 × PAGESIZE) to ((5 × PAGESIZE) − 0x4)	SMMU SSD address space	PAGESIZE global address space, Secure only
(5 × PAGESIZE) to ((6 × PAGESIZE) − 0x4)	In SMMUv2, continuation of SMMU address space. In SMMUv1, reserved.	-
(6 × PAGESIZE) to ((7 × PAGESIZE) − 0x4)	If SMMU_IDR0.EXSMRGS[a] is 1, SMMU_EXSMRn. Otherwise, IMPLEMENTATION DEFINED.	For more information, see *SMMU_EXSMRn, Extended Stream Match Register on page 14-279*
(7 × PAGESIZE) to ((8 × PAGESIZE) − 0x4)	If SMMU_IDR0.EXSMRGS[a] is 1, SMMU_EXS2CRn. Otherwise, IMPLEMENTATION DEFINED.	For more information, see *SMMU_EXS2CRn, Extended Stream-to-Context Register on page 14-280*
(8 × PAGESIZE) to ((23 × PAGESIZE) − 0x4)	If SMMU_IDR2.COMPINDEXS[b] is 1, SMMU_COMPINDEXn. Otherwise, IMPLEMENTATION DEFINED.	For more information, see *SMMU_COMPINDEXn, Compressed Index register n on page 15-284*
(24 × PAGESIZE) to SMMU_GLOBAL_TOP	IMPLEMENTATION DEFINED.	-

a. As returned by a Secure read of SMMU_IDR0, see the register description for more information. Indicates that the Extended Stream Match Extension is implemented.

b. As returned by a Secure read of SMMU_IDR2, see the register description for more information. Indicates that the StreamID Compressed Indexing Extension is implemented. Depending on the implemented range of StreamIDs, up to 2^{16} SMMU_COMPINDEXn registers are implemented. For this maximum number of registers, if PAGESIZE is 4KB they occupy all 16 pages of this allocation. However, if PAGESIZE is 64KB they occupy only the first page, and the other pages are reserved.

8.2.1 Global address space base addresses

The global address space base addresses defined in Table 8-2 are referred to in subsequent sections.

Table 8-2 Global address space base addresses

Base address name	Description	Value
SMMU_GR0_BASE	Base address of SMMU Global Register Space 0	SMMU_BASE + (0 × PAGESIZE)
SMMU_GR1_BASE	Base address of SMMU Global Register Space 1	SMMU_BASE + (1 × PAGESIZE)
SMMU_GID_BASE	Base address of SMMU IMPLEMENTATION DEFINED address space	SMMU_BASE + (2 × PAGESIZE)
SMMU_PM_BASE	Base address of SMMU performance monitoring address space	SMMU_BASE + (3 × PAGESIZE)
SMMU_SSD_BASE	Base address of SMMU SSD address space	SMMU_BASE + (4 × PAGESIZE)

For more information, see:

- Chapter 9 *SMMU Global Register Space 0*.
- Chapter 10 *SMMU Global Register Space 1*.
- Chapter 11 *SMMU IMPLEMENTATION DEFINED Address Space*.
- Chapter 12 *SMMU Performance Monitors Extension Register Map*.
- Chapter 13 *The Security State Determination Address Space*.

8.3 The translation context bank address space

Table 8-3 shows how the SMMU translation context bank address space is organized, relative to the SMMU context bank base address, SMMU_CB_BASE.

Table 8-3 Translation context bank address space

Offset	Description	Notes
`0x00000` to ((1 × PAGESIZE) − `0x4`)	Translation context bank 0	PAGESIZE per translation context bank
(1 × PAGESIZE) to ((2 × PAGESIZE) − `0x4`)	Translation context bank 1	-
(*n* × PAGESIZE) to ((*n* + 1 × PAGESIZE) − `0x4`)	Translation context bank *n*	-

SMMU_IDR1.NUMCB specifies the IMPLEMENTATION DEFINED number of translation context banks.

In an implementation that supports stage 1 followed by stage 2 translation, some translation context banks must be configured as stage 1 translation contexts, and the remaining translation context banks as stage 2 translation contexts. Because of the increased state requirements of stage 1 translation contexts, some translation contexts might only support stage 2 translation. SMMU_IDR1.NUMS2CB provides more information about discovering whether an implementation uses such contexts.

Any address above the upper implemented translation context bank address and below SMMU_TOP behaves as RAZ/WI.

Each translation context bank is defined in terms of its Context Bank Base Address, SMMU_CBn_BASE. The base address for translation context bank *n* is defined as SMMU_CBn_BASE = SMMU_CB_BASE + (*n* × PAGESIZE).

For more information, see:

- Chapter 16 *Stage 1 Translation Context Bank Format*.
- Chapter 17 *Stage 2 Translation Context Bank Format*.

Chapter 9
SMMU Global Register Space 0

This chapter specifies SMMU Global Register Space 0. It contains the following sections:

9.1 SMMU Global Register Space 0 register summary

SMMU Global Register Space 0 provides high-level SMMU resource control. The size of this address space is defined by PAGESIZE. See *PAGESIZE and NUMPAGENDXB on page 8-164* for more information.

The SMMU architecture supports atomic register access at the size of the register. Whether atomic register accesses of other sizes are supported is IMPLEMENTATION DEFINED.

Table 9-1 shows the registers in this register space, in order of their address offsets from SMMU_GR0_BASE. Unless otherwise stated, registers are present in any version of the SMMU architecture.

Table 9-1 SMMU Global Register Space 0

Offset	Name	Type	Width	Description	Notes
0x000	SMMU_sCR0	RW	32	*SMMU_sCR0, Configuration Register 0 on page 9-189*	Banked by security
0x004	SMMU_SCR1	RW	32	*SMMU_SCR1, Secure Configuration Register 1 on page 9-226*	Secure only
0x008	SMMU_sCR2	RW	32	*SMMU_sCR2, Configuration Register 2 on page 9-200*	Banked by security
0x00C	Reserved	-	-	-	-
0x010	SMMU_sACR	RW	32	*SMMU_sACR, Auxiliary Configuration Register on page 9-189*	Banked by security
0x014 - 0x01C	Reserved	-	-	-	-
0x020	SMMU_IDR0	RO	32	*SMMU_IDR0 on page 9-182*	-
0x024	SMMU_IDR1	RO	32	*SMMU_IDR1 on page 9-184*	-
0x028	SMMU_IDR2	RO	32	*SMMU_IDR2 on page 9-186*	-
0x02C	SMMU_IDR3	RO	32	*SMMU_IDR3 on page 9-189*	-
0x030, 0x034	SMMU_IDR4, SMMU_IDR5	RO	32	*SMMU_sIDR4-5 on page 9-189*	-
0x038	SMMU_IDR6	RO	32	*SMMU_IDR6 on page 9-189*	-
0x03C	SMMU_IDR7	RO	32	*SMMU_IDR7 on page 9-189*	-
0x040	SMMU_sGFAR	RW	64	*SMMU_sGFAR, Global Fault Address Register on page 9-208*	Banked by security
0x048	SMMU_sGFSR	RW	32	*SMMU_sGFSR, Global Fault Status Register on page 9-209*	Banked by security
0x04C	SMMU_sGFSRRESTORE	WO	32	*SMMU_sGFSRRESTORE, Global Fault Status Restore Register on page 9-211*	Banked by security
0x050	SMMU_sGFSYNR0	RW	32	*SMMU_sGFSYNR0, Global Fault Syndrome Register 0 on page 9-211*	Banked by security
0x054	SMMU_sGFSYNR1	RW	32	*SMMU_sGFSYNR1, Global Fault Syndrome Register 1 on page 9-213*	Banked by security
0x058	SMMU_sGFSYNR2	RW	32	*SMMU_sGFSYNR2, Global Fault Syndrome Register 2 on page 9-214*	Banked by security

Table 9-1 SMMU Global Register Space 0 (continued)

Offset	Name	Type	Width	Description	Notes
0x05C	Reserved	-	-	-	-
0x060	SMMU_STLBIALL	WO	32	*SMMU_STLBIALL, TLB Invalidate All on page 9-232*	Secure only
0x064	SMMU_TLBIVMID	WO	32	*SMMU_TLBIVMID, TLB Invalidate by VMID on page 9-236*	Banked by security
0x068	SMMU_TLBIALLNSNH	WO	32	*SMMU_TLBIALLNSNH, TLB Invalidate All Non-secure Non-Hyp on page 9-233*	Banked by security
0x06C	SMMU_TLBIALLH	WO	32	*SMMU_TLBIALLH, TLB Invalidate All Hyp on page 9-233*	Banked by security
0x070	SMMU_sTLBGSYNC	WO	32	*SMMU_sTLBGSYNC, Global Synchronize TLB Invalidate on page 9-219*	Banked by security
0x074	SMMU_sTLBGSTATUS	RO	32	*SMMU_sTLBGSTATUS, Global TLB Status register on page 9-218*	Banked by security
0x078	SMMU_TLBIVAH	WO	32	*SMMU_TLBIVAH, Invalidate Hyp TLB by VA on page 9-235*	Banked by security
0x07C	Reserved	-	-	-	-
0x080 - 0x09C	IMPLEMENTATION DEFINED	-	-	Reserved for TLB Debug features	-
0x0A0	SMMU_STLBIVALM	WO	64	*SMMU_STLBIVALM, Secure TLB Invalidate MONC by VA, Last level on page 9-233*	SMMUv2 only Secure only
0x0A8	SMMU_STLBIVAM	WO	64	*SMMU_STLBIVAM, Secure TLB Invalidate MONC by VA on page 9-234*	SMMUv2 only Secure only
0x0B0	SMMU_TLBIVALH64	WO	64	*SMMU_TLBIVALH64, Invalidate Hyp TLB by VA, Last level, AArch64 on page 9-236*	SMMUv2 only. Banked by security.
0x0B8	SMMU_TLBIVMIDS1	WO	32	*SMMU_TLBIVMIDS1, TLB Invalidate Stage 1 by VMID on page 9-237*	SMMUv2 only. Banked by security.
0x0BC	SMMU_STLBIALLM	WO	32	*SMMU_STLBIALLM, Secure TLB Invalidate All MONC on page 9-233*	SMMUv2 only Secure only
0x0C0	SMMU_TLBIVAH64	WO	64	*SMMU_TLBIVAH64, Invalidate Hyp TLB by VA, AArch64 on page 9-235*	SMMUv2 only. Banked by security
0x0C8- 0x0FC	Reserved	-	-	-	-
0x100	SMMU_sGATS1UR	WO	64	*SMMU_sGATS1UR, GAT Stage 1 Unprivileged Read on page 9-203*	Banked by security
0x108	SMMU_sGATS1UW	WO	64	*SMMU_sGATS1UW, GAT Stage 1 Unprivileged Write on page 9-204*	Banked by security
0x110	SMMU_sGATS1PR	WO	64	*SMMU_sGATS1PR, GAT Stage 1 Privileged Read on page 9-202*	Banked by security

Table 9-1 SMMU Global Register Space 0 (continued)

Offset	Name	Type	Width	Description	Notes
0x118	SMMU_sGATS1PW	WO	64	*SMMU_sGATS1PW, GAT Stage 1 Privileged Write on page 9-202*	Banked by security
0x120	SMMU_sGATS12UR	WO	64	*SMMU_sGATS12UR, GAT Stages 1 and 2 Unprivileged Read on page 9-206*	Banked by security
0x128	SMMU_sGATS12UW	WO	64	*SMMU_sGATS12UW, GAT Stages 1 and 2 Unprivileged Write on page 9-207*	Banked by security
0x130	SMMU_sGATS12PR	WO	64	*SMMU_sGATS12PR, GAT Stages 1 and 2 Privileged Read on page 9-205*	Banked by security
0x138	SMMU_sGATS12PW	WO	64	*SMMU_sGATS12PW, GAT Stages 1 and 2 Privileged Write on page 9-205*	Banked by security
0x140-0x17C	Reserved	-	-	-	-
0x180	SMMU_sGPAR	RW	64	*SMMU_sGPAR, Global Physical Address Register on page 9-214*	Banked by security
0x188	SMMU_sGATSR	RO	32	*SMMU_sGATSR, Global Address Translation Status Register on page 9-208*	Banked by security
0x18C-0x3FC	Reserved	-	-	-	-
0x400	SMMU_NSCR0	RW	32	Secure alias for Non-secure SMMU_CR0	Secure only
0x404	Reserved	-	-	-	-
0x408	SMMU_NSCR2	RW	32	Secure alias for Non-secure copy of *SMMU_sCR2, Configuration Register 2 on page 9-200*	Secure only
0x40C	Reserved	-	-	-	-
0x410	SMMU_NSACR	RW	32	Secure alias for Non-secure copy of *SMMU_sACR, Auxiliary Configuration Register on page 9-189*	Secure only
0x414-0x41C	Reserved	-	-	-	-
0x420 - 0x43C	Reserved	-	-	-	-
0x440	SMMU_NSGFAR	RW	64	Secure alias for Non-secure copy of *SMMU_sGFAR, Global Fault Address Register on page 9-208*	Secure only
0x448	SMMU_NSGFSR	RW	32	Secure alias for Non-secure copy of *SMMU_sGFSR, Global Fault Status Register on page 9-209*	Secure only
0x44C	SMMU_NSGFSRRESTORE	WO	32	Secure alias for Non-secure copy of *SMMU_sGFSRRESTORE, Global Fault Status Restore Register on page 9-211*	Secure only

Table 9-1 SMMU Global Register Space 0 (continued)

Offset	Name	Type	Width	Description	Notes
0x450	SMMU_NSGFSYNR0	RW	32	Secure alias for Non-secure copy of *SMMU_sGFSYNR0, Global Fault Syndrome Register 0 on page 9-211*	Secure only
0x454	SMMU_NSGFSYNR1	RW	32	Secure alias for Non-secure copy of *SMMU_sGFSYNR1, Global Fault Syndrome Register 1 on page 9-213*	Secure only
0x458	SMMU_NSGFSYNR2	RW	32	Secure alias for Non-secure copy of *SMMU_sGFSYNR2, Global Fault Syndrome Register 2 on page 9-214*	Secure only
0x45C - 0x46C	Reserved	-	-	-	-
0x470	SMMU_NSTLBGSYNC	WO	32	Secure alias for Non-secure copy of *SMMU_sTLBGSYNC, Global Synchronize TLB Invalidate on page 9-219*	Secure only
0x474	SMMU_NSTLBGSTATUS	RO	32	Secure alias for Non-secure copy of *SMMU_sTLBGSTATUS, Global TLB Status register on page 9-218*	Secure only
0x478 - 0x47C	Reserved	-	-	-	-
0x480 - 0x49C	IMPLEMENTATION DEFINED	-	-	Reserved for Secure alias to Non-secure TLB debug features	Secure only
0x4A0- 0x4FC	Reserved	-	-	-	-
0x500	SMMU_NSGATS1UR	WO	64	Secure alias for Non-secure copy of *SMMU_sGATS1UR, GAT Stage 1 Unprivileged Read on page 9-203*	Secure only
0x508	SMMU_NSGATS1UW	WO	64	Secure alias for Non-secure copy of *SMMU_sGATS1UW, GAT Stage 1 Unprivileged Write on page 9-204*	Secure only
0x510	SMMU_NSGATS1PR	-	64	Secure alias for Non-secure copy of *SMMU_sGATS1PR, GAT Stage 1 Privileged Read on page 9-202*	Secure only
0x518	SMMU_NSGATS1PW	WO	64	Secure alias for Non-secure copy of *SMMU_sGATS1PW, GAT Stage 1 Privileged Write on page 9-202*	Secure only
0x520	SMMU_NSGATS12UR	WO	64	Secure alias for Non-secure copy of *SMMU_sGATS12UR, GAT Stages 1 and 2 Unprivileged Read on page 9-206*	Secure only
0x528	SMMU_NSGATS12UW	WO	64	Secure alias for Non-secure copy of *SMMU_sGATS12UW, GAT Stages 1 and 2 Unprivileged Write on page 9-207*	Secure only
0x530	SMMU_NSGATS12PR	WO	64	Secure alias for Non-secure copy of *SMMU_sGATS12PR, GAT Stages 1 and 2 Privileged Read on page 9-205*	Secure only

Table 9-1 SMMU Global Register Space 0 (continued)

Offset	Name	Type	Width	Description	Notes
0x538	SMMU_NSGATS12PW	WO	64	Secure alias for Non-secure copy of *SMMU_sGATS12PW, GAT Stages 1 and 2 Privileged Write on page 9-205*	Secure only
0x540 - 0x57C	Reserved	-	-	-	-
0x580	SMMU_NSGPAR	RW	64	Secure alias for Non-secure copy of *SMMU_sGPAR, Global Physical Address Register on page 9-214*	Secure only
0x588	SMMU_NSGATSR	RO	32	Secure alias for Non-secure copy of *SMMU_sGATSR, Global Address Translation Status Register on page 9-208*	Secure only
0x58C - 0x7FC	Reserved	-	-	-	-
0x800	SMMU_SMR0	RW	32	*SMMU_SMRn, Stream Match Register on page 9-231*	RAZ/WI in an implementation with StreamID indexing
0x804	SMMU_SMR1				
0x808-0x9FC	SMMU_SMR2 to SMMU_SMR127				
0xA00-0xBFC	Reserved	-		-	-
0xC00	SMMU_S2CR0	RW	32	*SMMU_S2CRn, Stream-to-Context Register on page 9-220*	-
0xC04	SMMU_S2CR1				
0xC08-0xDFC	SMMU_S2CR2 to SMMU_S2CR127				
0xE00-0xFCC	Reserved	-	-	-	-
0xFD0-0xFFC	IMPLEMENTATION DEFINED	-	-	Can be used for AMBA ID registers, if appropriate	-

9.2 Reset values

Table 9-2 shows the register field values in SMMU Global Register Space 0 following a system reset. The fields that do not appear in Table 9-2 reset to UNKNOWN values.

Table 9-2 Reset values

Field	Reset	Effect of the reset value
SMMU_CBA2Rn.VA64	IMPLEMENTATION DEFINED	In SMMUv2, disable AArch64 translation scheme. This bit becomes UNKNOWN when a context bank changes Security state, see *Resource allocation on page 7-158*.
SMMU_sCR0.CLIENTPD	0b1	Client transaction bypasses SMMU translation
SMMU_sCR0.GFIE	0b0	For SMMU_CR0.GFIE, disable assertion of interrupt from UNPREDICTABLE SMMU_GFSR content. For SMMU_SCR0.GFIE, disable assertion of interrupt from UNPREDICTABLE SMMU_SGFSR content. See *SMMU_sGFSR, Global Fault Status Register on page 9-209*.
SMMU_sCR0.GFRE	0b0	For SMMU_CR0.GFRE, disable spurious abort from UNPREDICTABLE SMMU_GFSR content. For SMMU_SCR0.GFRE, disable spurious abort from UNPREDICTABLE SMMU_SGFSR content. See *SMMU_sGFSR, Global Fault Status Register on page 9-209*.
SMMU_sCR0.GCFGFIE	0b0	For SMMU_CR0.GCFGFIE, disable assertion of interrupt from UNPREDICTABLE SMMU_GFSR content. For SMMU_SCR0.GCFGFIE, disable assertion of interrupt from UNPREDICTABLE SMMU_SGFSR content. See *SMMU_sGFSR, Global Fault Status Register on page 9-209*.
SMMU_sCR0.GCFGFRE	0b0	For SMMU_CR0.GCFGFRE, disable spurious abort from UNPREDICTABLE SMMU_GFSR content. For SMMU_SCR0.GCFGFRE, disable spurious abort from UNPREDICTABLE SMMU_SGFSR content. See *SMMU_sGFSR, Global Fault Status Register on page 9-209*.
SMMU_sCR0.GSE	0b0	Disable global stalling across translation context banks
SMMU_sCR0.MTCFG	0b0	No override of input memory attributes
SMMU_sCR0.SHCFG	0b00	No override of input shareability attributes
SMMU_sCR0.RACFG	0b00	No override of input read allocate attribute
SMMU_sCR0.WACFG	0b00	No override of input write allocate attribute
SMMU_sCR0.TRANSIENTCFG	0b00	No override of input transient allocate attribute
SMMU_sCR0.EXIDENABLE	0b0	Disables use of the Extended StreamID format
SMMU_sCR0.STALLD	0b0	If stalling is supported, then must reset to 0b0 permitting per context stalling on context faults. If stalling is not supported, this resets to an UNKNOWN value.
SMMU_sCR0.NSCFG	0b00	No override of input NS-Attr
SMMU_sTLBGSTATUS.GSACTIVE	IMPLEMENTATION DEFINED	No active Global TLB Invalidate operation Sync
SMMU_SCR1.NSCAFRO	IMPLEMENTATION DEFINED	This bit may be a fixed IMPLEMENTATION DEFINED value, or if programmable, resets to 0

Table 9-2 Reset values (continued)

Field	Reset	Effect of the reset value
SMMU_SCR1.GEFRO	0b0	-
SMMU_SCR1.NSNUMCBO	Implemented NUMCB	Do not require Secure software to write NSNUMCBO if such software is not using any SMMU context
SMMU_SCR1.GASRAE	0b0	Permit Secure and Non-secure configuration accesses
SMMU_SCR1.SPMEN	0b0	No contribution to performance monitor counters from any Secure transaction processing event
SMMU_SCR1.SIF	0b0	Secure state instruction fetch permitted to Non-secure memory.
SMMU_SCR1.NSNUMIRPTO	Implemented NUMIRPT	It is IMPLEMENTATION DEFINED whether this field is read-only. See *SMMU_SCR1, Secure Configuration Register 1 on page 9-226*.
SMMU_SCR1.NSNUMSMRGO	Implemented NUMSMRG	-

9.3 Secure alias for Non-secure registers

In an implementation that supports two Security states, additional alias registers are provided at different offset addresses, permitting Secure software to access Non-secure versions of the resources.

In general, the aliases are accessed in the following usage cases:

Secure software manages Non-secure context resources

When the SMMU_SCR1.GASRAE bit is set to 1, the Global address space is only accessible by Secure configuration memory accesses. This means that Non-secure software must request that Secure software performs certain operations. In such cases it might be necessary for Secure software to access any of:

- SMMU_NSCR0.
- SMMU_NSCR2.
- SMMU_NSACR.
- SMMU_NSGFAR.
- SMMU_NSGFSR.
- SMMU_NSGFSRRESTORE.
- SMMU_NSGFSYNRn.
- SMMU_NSGPAR.
- SMMU_NSGATSR.
- SMMU_NSGATS1*.

For example, a Secure access to SMMU_NSGFSYNR1 is the only mechanism for obtaining the SSD_Index for a faulty Non-secure transaction.

The SMMU architecture provides separate Secure and Non-secure address translation resources. If Secure software is managing the Non-secure resources, it can use the Non-secure address translation registers, and the Secure resources remain available for Secure software to use as required.

Note

Although the SMMU architecture provides separate Secure and Non-secure TLB Invalidation resources, Secure software cannot initiate a TLB Invalidate operation directly using Non-secure resources. In this usage case, Secure software must use Secure resources to initiate these operations.

Save and restore, or Powerdown operations

During save and restore operations, such as before Powerdown, Secure software can save the Non-secure state by accessing the following aliases:

- SMMU_NSCR0.
- SMMU_NSCR2.
- SMMU_NSACR.
- SMMU_NSGFAR.
- SMMU_NSGFSR.
- SMMU_NSGFSRRESTORE.
- SMMU_NSGFSYNRn.
- SMMU_NSGPAR.

During Powerdown, Secure software must ensure that the SMMU is quiescent by:

- Completing all TLB Invalidate operations.
- Completing all address translation operations.
- Ensuring that all upstream devices are quiescent.

To ensure that any queued Non-secure TLB Invalidate operations are issued and completed, Secure software accesses SMMU_NSTLBGSYNC to start the operations, then polls SMMU_NSTLBGSTATUS to check for completion. Similarly, Secure software polls SMMU_NSGATSR to ensure that all Non-secure address translation operations are complete.

Note

When Secure software issues an SMMU_TLBIALLNSNH or SMMU_TLBIALLH operation, it uses Secure TLB Invalidate resources, and manages them using SMMU_STLBGSYNC and SMMU_STLBGSTATUS. This is distinct from a requirement to use SMMU_NSTLBGSYNC or SMMU_NSTLBGSTATUS. In general, the Secure alias must only be used when an operation is not otherwise possible.

9.4 Memory attribute, MemAttr

The MemAttr field is common to a number of registers. MemAttr[3:2] gives a top-level definition of the memory type, and of the cacheability of a Normal memory region. See Table 9-3.

The encoding of MemAttr[1:0] depends on the memory type indicated by MemAttr[3:2]:

- When MemAttr[3:2] == `0b00`, indicating Strongly-ordered or Device memory, Table 9-4 shows the encoding of MemAttr[1:0].
- When MemAttr[3:2] != `0b00`, indicating Normal memory, Table 9-5 shows the encoding of MemAttr[1:0].

Table 9-3 MemAttr[3:2] encoding

MemAttr[3:2]	Memory type	Cacheability
`0b00`	Strongly-ordered or Device, determined by MemAttr[1:0]	Not applicable
`0b01`	Normal	Outer Non-cacheable
`0b10`		Outer Write-Through Cacheable
`0b11`		Outer Write-Back Cacheable

Table 9-4 MemAttr[1:0] encoding for Strongly-ordered or Device memory

MemAttr[1:0]	Meaning when MemAttr[3:2] == `0b00`	
	SMMUv2	SMMUv1
`0b00`	Device-nGnRnE	Strongly-ordered memory
`0b01`	Device-nGnRE	Device memory
`0b10`	Device-nGRE	UNPREDICTABLE
`0b11`	Device-GRE	UNPREDICTABLE

Table 9-5 MemAttr[1:0] encoding for Normal memory

MemAttr[1:0]	Meaning when MemAttr[3:2] != `0b00`	
	SMMUv2	SMMUv1
`0b00`	Reserved. See *Reserved memory type and shareability attributes on page 2-62*.	UNPREDICTABLE
`0b01`	Inner Non-cacheable	Inner Non-cacheable
`0b10`	Inner Write-Through Cacheable	Inner Write-Through Cacheable
`0b11`	Inner Write-Back Cacheable	Inner Write-Back Cacheable

9.5 Multi-format registers and reserved fields

Some registers have a number of possible formats, where the format depends on:

- The format that the register has been configured to.
- The format of the data recorded in the register.

Several register formats have a TYPE field, that has different encodings for different registers, and some registers are written with data whose format depends on the outcome of the operation for which the result is to be written.

For such a register, where a field is reserved:

- A read must return the value that was last successfully written to that field, regardless of how the register is used. If the field has not been written to since reset, the read must return the specified reset value, if one exists, or an UNKNOWN value. See *Reset values on page 9-175*.
- A write must update a storage location associated with the written field.
- The state of the storage location associated with a field must have no other effect on the processor behavior, other than determining the value to be read back while the use of the register means that the field is reserved.

If the TYPE field changes so that a reserved field becomes a field with defined behavior, the value last written to that field takes effect as specified by the individual register descriptions.

Each behavior only applies to a field that is:

- Reserved in one format with an allocated meaning in a different format.
- Being processed by hardware.

A field processed by software is to be preserved if appropriate, or written to with the value required by the *should be* value if being written with a new value. Otherwise, the effect is UNPREDICTABLE.

9.6 SMMU Global Register Space 0 register descriptions

This section describes all of the Global Register Space 0 registers that might be present in an SMMU implementation. Unless otherwise stated, registers are present in any version of the SMMU architecture. Registers are shown in register name order.

- *Register names on page xiii*.
- *Banked registers on page 2-51*.

9.6.1 SMMU_IDR0-7, Identification registers

The SMMU_IDR0-7 characteristics are:

Purpose Provides SMMU capability information.

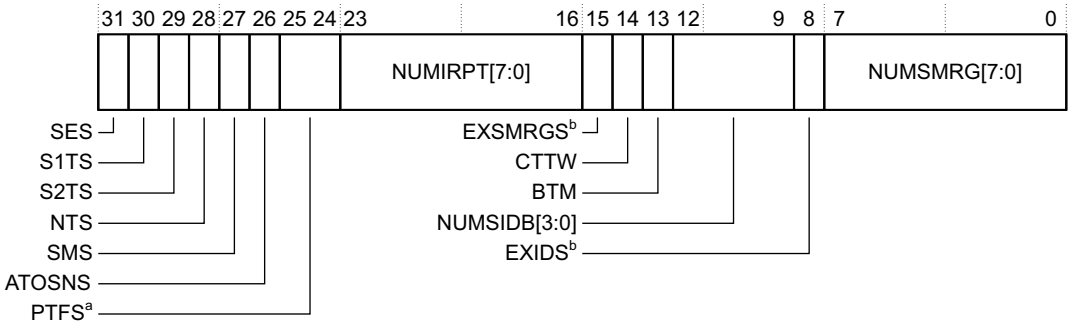
Usage constraints There are no usage constraints.

Configurations In an implementation that supports two Security states, the behavior of some fields depends on whether Secure or Non-secure software is accessing the register. See the field descriptions for more information.

Attributes 32-bit RO registers with UNKNOWN reset values. See also Table 9-1 on page 9-170.

SMMU_IDR0

The SMMU_IDR0 bit assignments are:



a. In SMMUv1, bit[25] is reserved, and bit[24] is PTFS
b. Reserved in SMMUv1

SES, bit[31] Security Support. The possible values of this bit are:

0 The implementation does not support two Security states.

1 The implementation supports two Security states.

This field is RAZ for Non-secure reads of SMMU_IDR0.

S1TS, bit[30] Stage 1 Translation Support. The possible values of this bit are:

0 Stage 1 translation is not supported.

1 Stage 1 translation is supported.

If the NTS bit is set to 1, this bit must also be set to 1.

S2TS, bit[29] Stage 2 Translation Support. The possible values of this bit are:

0 Stage 2 translation is not supported.

1 Stage 2 translation is supported.

This field only applies to Non-secure client transactions.

If the NTS bit is set to 1, this bit must also be set to 1.

NTS, bit[28] Nested Translation Support. The possible values of this bit are:

0 Stage 1 followed by stage 2 translation is not supported.

1 Stage 1 followed by stage 2 translation is supported.

This field only applies to Non-secure client transactions.

Note

If this bit is set to 1, then both the S1TS and S2TS bits must be set to 1 also.

SMS, bit[27] Stream Match Support. The possible values of this bit are:

0 Stream Match Register functionality is not included.

1 Stream Match Register functionality is included.

When Stream matching is supported, Stream indexing is not supported. Otherwise, Stream indexing is supported.

ATOSNS, bit[26]

Address Translation Operations Not Supported. The possible values of this bit are:

0 Address translation operations are supported.

1 Address translation operations are not supported.

This bit is valid only when stage 1 translations are supported, as indicated by the value of SMMU_IDR0.S1TS being 1. When SMMU_IDR0.S1TS is 0, this bit is 0.

In SMMUv1, this bit is reserved.

See *Address translation registers in SMMUv2 on page 4-126* for more information.

PTFS, bits[25:24]

AArch32 translation formats support. This field indicates the supported translation schemes, and therefore the translation table format. The possible values of this field are:

`0b00` AArch32 Short-descriptor and AArch32 Long-descriptor translation schemes are supported.

`0b01` Only AArch32 Long-descriptor translation scheme is supported.

`0b10` No AArch32 translation formats are supported.

`0b11` Reserved.

Note

- In SMMUv1, bit[25] is reserved, and PTFS is bit[24], with possible values 0 and 1.
- In SMMUv2, the AArch64 translation formats supported are indicated by the SMMU_IDR2 PTFSv8_* fields.

NUMIRPT[7:0], bits[23:16]

Number of implemented context fault interrupts.

In SMMUv1, this field indicates the number of context fault interrupts supported by the SMMU, in the range 0-128. NUMIRPT==0 is permitted in an implementation that does not provide any context fault interrupts.

In any implementation that supports two Security states, access to this field by Non-secure software gives the value configured in SMMU_SCR1.NSNUMIRPTO.

In SMMUv2, each context bank has a dedicated interrupt pin, and there is therefore no requirement to specify the number of implemented context fault interrupts. See *Context interrupts on page 3-100* for more information.

EXSMRGS, bit[15]

In SMMUv2, if the Extended Stream Matching extension is implemented then:

- On a Secure register access, this bit returns 1.
  - Returns 0 if SMMU_SCR2.EXNSSMRGDISABLE is 1.
  - Returns 1 if SMMU_SCR2.EXNSSMRGDISABLE is 0.

In SMMUv2 if the Extended Stream Matching extension is not implemented, and in SMMUv1, this bit is RES0.

CTTW, bit[14] Coherent Translation Table Walk. The possible values of this bit are:

0 Coherent translation table walks are unsupported.

1 Coherent translation table walks are supported.

BTM, bit[13] Broadcast TLB Maintenance. The possible values of this bit are:

0 Broadcast TLB maintenance is unsupported.

1 Broadcast TLB maintenance is supported.

NUMSIDB[3:0], bits[12:9]

Number of StreamID Bits.

Indicates the number of implemented StreamID bits, in the range 0-15.

In SMMUv2, the SMMU_S2CRn.EXIDVALID bit provides optional support for 16-bit StreamIDs. When EXIDS==0, this field indicates the number of implemented StreamID bits, in the range 0-15. Otherwise this field is 15.

EXIDS, bit[8] Extended StreamID Support. The possible values of this bit are:

0 Extended StreamIDs are unsupported, and the NUMSIDB field indicates the maximum supported StreamID size.

1 Extended StreamIDs are supported. The number of supported StreamID bits is 16 and the NUMSIDB field is 15.

In SMMUv1, this bit is reserved.

NUMSMRG[7:0], bits[7:0]

Number of Stream Mapping Register Groups.

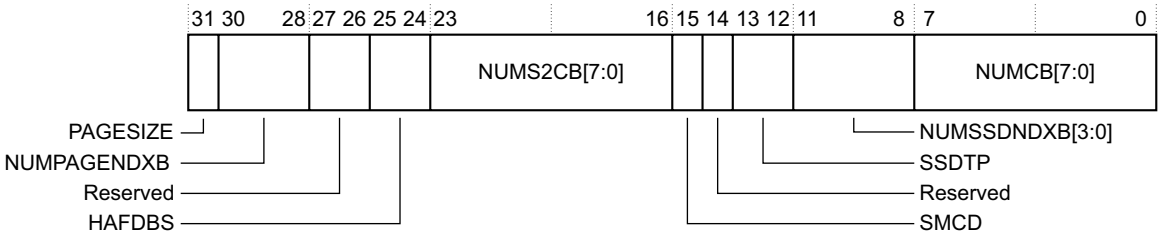
Indicates the number of Stream mapping register groups in the Stream match table, in the range 0-128.

In an implementation that supports two Security states, a Non-secure read of this field returns the value of SMMU_SCR1.NSNUMSMRGO.

When the Extended Stream Matching Extension is implemented, a Secure read of this field returns the value 128.

SMMU_IDR1

The SMMU_IDR1 bit assignments are:



In SMMUv1, bit[13] is reserved and bit[12] is SSDTP

PAGESIZE, bit[31]

SMMU Page Size.

Indicates the size of each page in the SMMU register map.

The possible values of this bit are:

0 4KB.

1 64KB.

NUMPAGENDXB, bits[30:28]

SMMU Number of Page Index Bits.

Indicates how many PAGESIZE pages occupy the global address space or the translation context address space, where `NUMPAGE = 2(SMMU_IDR.NUMPAGENDXB + 1)`.

Bits[27:26] Reserved.

HAFDBS, bits, bits[25:24]

Hardware Access Flag and Dirty Bit management supported. These bits are the stage 1 SMMU_CBn_TCR2.{HA, HD} and stage 2 SMMU_CBn_TCR.{HA, HD} bits.

The possible values of this field are:

00 No support for hardware update of either Access flag or Dirty bit information.

01 Support for hardware update of Access flag, but no support for hardware update of Dirty bit information.

10 Reserved.

11 Support for hardware update of both Access flag and Dirty bit information.

In SMMUv1, bits[25:24] are reserved.

NUMS2CB[7:0], bits[23:16]

Number of stage 2 Context Banks.

Indicates the number of translation context banks that only support the stage 2 translation format, in the range 0-128.

This field is validated by SMMU_IDR0.S2TS.

SMCD, bit[15] Stream Match Conflict Detection.

The possible values of this bit are:

0 The detection of all Stream match conflicts is not guaranteed.

1 The detection of all Stream match conflicts is guaranteed.

See *StreamID matching on page 2-57* for more information about stream matching.

Bit[14] Reserved.

SSDTP, bits[13:12]

SSD Table Present. The possible values of this field are:

`0b00` The SSD address space is UNK/WI.

`0b01` The SSD address space is populated and supports the SSD_Index size that NUMSSDNDXB specifies.

`0b10` Reserved.

`0b11` The extended SSD address space is present and supports a 16-bit SSD_Index.

Note

In SMMUv1, bit[13] is reserved, and SSDTP is bit[12], with possible values 0 and 1.

In an implementation that supports two Security states, Non-secure access to this field is RAZ.

NUMSSDNDXB[3:0], bits[11:8]

Number of SSD_Index bits.

Indicates the number of SSD_Index bits for indexing the SSD table.

In SMMUv2, if the SSDTP field is `0b11`, this field is `0b1111` for backwards compatibility

In SMMUv1, this field is only valid if SSDTP==1. Otherwise, it is reserved.

In an implementation that supports two Security states Non-secure access to this field is RAZ.

NUMCB, bits[7:0]

Number of Context Banks.

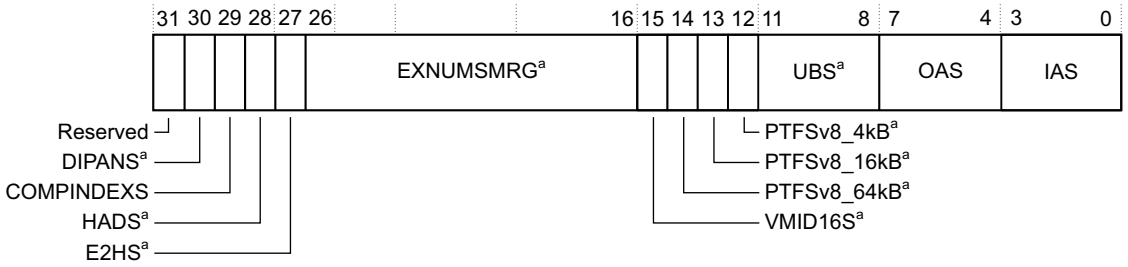
Indicates the total number of implemented translation context banks, in the range 0-128.

The value reported in NUMCB includes translation context banks that only support the stage 2 format. If an implementation includes stage 1 translation, the number of translation context banks that support the stage 1 format is given by SMMU_IDR1.NUMCB – SMMU_IDR1.NUMS2CB.

In an implementation that supports two Security states, a read of this field by Non-secure software gives the value configured in SMMU_SCR1.NSNUMCBO.

SMMU_IDR2

The SMMU_IDR2 bit assignments are:



a. Reserved in SMMUv1

Bit[31] Reserved.

DIPANS, bit[30]

Data and Instruction Privileged Access Never Support. The possible values of this bit are:

0 Privileged Access Never not supported.

1 Privileged Access Never supported.

When SMMU_IDR2.DIPANS is 1, the `0b01` encoding of the SMMU_S2CRn.PRIVCFG field is redefined as *Data and Instruction Privilege Access Never*, and:

- For a context bank that supports two levels of privilege, when the value of SMMU_S2CRn.PRIVCFG is `0b01`, all transactions are treated as privileged, and a permission fault is generated if a transaction attempts to access a memory region that is accessible by unprivileged accesses.
- If a context bank is configured as Bypass, the SMMU_S2CRn.PRIVCFG encoding `0b01` has the same effect as encoding `0b11`, Privileged.
- HYPC and MONC banks do not distinguish between privileged and unprivileged accesses, and therefore for these banks the SMMU_S2CRn.PRIVCFG encoding `0b01` has the same effect as encoding `0b11`.

Note

The SMMU architecture differs from the PE architecture in that Privileged Access Never applies to both instruction and data accesses. For this reason, the SMMU control bit is called DIPANS.

This bit is RES0 in SMMUv1.

COMPINDEXS. bit[29]

StreamID Compressed Indexing Support. The possible values of this bit are:

0 StreamID compressed indexing not supported.

1 StreamID compressed indexing supported.

In SMMUv1, this bit is RES0.

In SMMUv2, this bit indicates support for the StreamID Compressed Indexing Extension. For more information, see Chapter 15 *StreamID Compressed Indexing Extension*.

When the StreamID Compressed Indexing Extension is implemented, for Non-secure accesses to this register, this field reads as zero if SMMU_SCR1.NSCOMPINDEXDISABLE == 1, indicating that the StreamID Compressed Indexing Extension is accessible to Secure state, and not accessible to Non-secure state.

HADS, bit[28]

Hierarchical Attribute Disable Support. The possible values of this bit are:

0 HAD not supported.

1 HAD supported. The stage 1 SMMU_CBn_TCR2.{HAD0, HAD1} bits are defined as for ARMv8.1 processor architecture.

This control applies only to the AArch64 translation scheme.

This bit is RES0 in SMMUv1.

E2HS, bit[27]

E2H context (E2HC) supported. This context provides the translation regime defined by the ARMv8.1 Virtualization Host Exceptions. In SMMUv2, support for the E2H context is optional. The possible values of this bit are:

0 E2HC not supported.

1 E2HC supported.

See *E2H contexts (E2HC) on page 2-90* for more information.

This bit is RES0 in SMMUv1.

EXNUMSMRG, bits[26:16]

For Secure accesses, the value returned by a read of this field is the number of Extended Stream Match Register groups implemented.

For Non-secure accesses, the values returned by a read of this field is the number of Extended Stream Match Register groups that can be used by Non-secure contexts. For more information, see *The number of Extended Stream Match Register groups available to Non-secure transactions on page 14-277*.

Note

When the Extended Stream Matching Extension is implemented, the number of Extended Stream Match Register groups must include the 128 groups that the {SMMU_SMRn, SMMU_S2CRn} register groups map onto.

This bit is RES0 in SMMUv1.

For implementations not supporting Extended Stream Match Register groups, this field is RAZ.

VMID16S, bit[15]

Support for 16-bit VMIDs. The possible values are:

0 Only 8-bit VMIDs are supported.

1 16-bit VMIDs are supported.

See *VMID size on page 1-39*.

This bit is RES0 in SMMUv1.

PTFSv8_64kB, bit[14]

Support for 64KB translation granule size. The possible values of this bit are:

0 The 64KB translation granule is not supported.

1 The 64KB translation granule is supported.

In SMMUv1, this bit is reserved.

PTFSv8_16kB, bit[13]

Support for 16KB translation granule size. The possible values of this bit are:

0 The 16KB translation granule is not supported.

1 The 16KB translation granule is supported.

In SMMUv1, this bit is reserved.

PTFSv8_4kB, bit[12]

Support for 4KB translation granule size. The possible values of this bit are:

0 The 4KB translation granule is not supported.

1 The 4KB translation granule is supported.

In SMMUv1, this bit is reserved.

UBS, bits[11:8]

Upstream Bus Size. The encoding of this field is:

`0b0000` 32-bit upstream bus size.

`0b0001` 36-bit upstream bus size.

`0b0010` 40-bit upstream bus size.

`0b0011` 42-bit upstream bus size.

`0b0100` 44-bit upstream bus size.

`0b0101` 49-bit upstream bus size.

`0b1111` 64-bit upstream bus size. See *Sign-extension of input addresses on page 1-38* for more information.

All other encodings are reserved.

In SMMUv1, this field is reserved.

OAS, bits[7:4] Output Address Size. This specifies the maximum number of PA bits that an SMMU implementation supports:

`0b0000` 32-bit.

`0b0001` 36-bit.

`0b0010` 40-bit.

`0b0011` 42-bit.

`0b0100` 44-bit.

`0b0101` 48-bit.

All other encodings are reserved.

IAS, bits[3:0] IPA Address Size. This specifies the maximum number of IPA bits that an SMMU implementation supports:

`0b0000` 32-bit.

`0b0001` 36-bit.

`0b0010` 40-bit.

`0b0011` 42-bit.

`0b0100` 44-bit.

`0b0101` 48-bit.

All other encodings are reserved.

SMMU_IDR3

The SMMU_IDR3 bit assignments are reserved.

SMMU_sIDR4-5

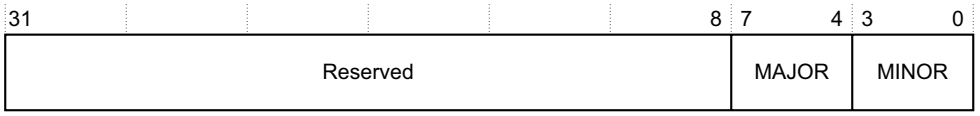
The SMMU_IDR4-5 bit assignments are IMPLEMENTATION DEFINED.

SMMU_IDR6

The SMMU_IDR6 bit assignments are reserved.

SMMU_IDR7

The SMMU_IDR7 bit assignments are:



Bits[31:8] Reserved.

MAJOR, bits[7:4] The major part of the implementation version number.

MINOR, bits[3:0] The minor part of the implementation version number.

9.6.2 SMMU_sACR, Auxiliary Configuration Register

The SMMU_*s*ACR characteristics are:

Purpose Provides IMPLEMENTATION DEFINED functionality.

Usage constraints There are no usage constraints.

Configurations In an implementation that supports two Security states, this register is Banked.

SMMU_NSACR is provided as a Secure alias of the Non-secure SMMU_ACR. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

Attributes A 32-bit RW register with an UNKNOWN reset value. See also Table 9-1 on page 9-170.

The SMMU_*s*ACR bit assignments are IMPLEMENTATION DEFINED.

9.6.3 SMMU_sCR0, Configuration Register 0

The SMMU_sCR0 characteristics are:

Purpose Provides top-level control of the SMMU.

Usage constraints In an implementation that supports two Security states, some fields are implemented only in the Secure copy of the register, SMMU_SCR0.

Configurations In an implementation that supports two Security states, the register is Banked to provide:

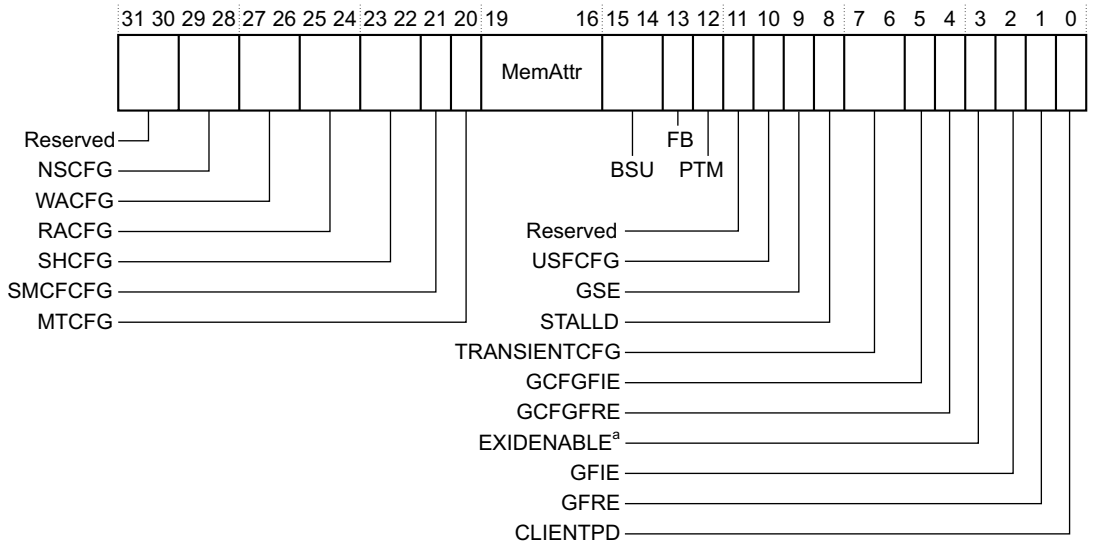
- SMMU_SCR0, the Secure Configuration Register 0, that provides configuration for Non-secure transactions, and some Secure state controls of Non-secure transactions.
- SMMU_CR0, the Non-secure Configuration Register 0, that applies only to Non-secure transactions.

Attributes A 32-bit RW register. See the field descriptions for information about the reset values. See also Table 9-1 on page 9-170. For the field descriptions, see:

- *SMMU_SCR0, Secure Configuration Register 0 bit assignments on page 9-190*.
- *SMMU_CR0, Non-secure Configuration Register 0 bit assignments on page 9-195*.

SMMU_SCR0, Secure Configuration Register 0 bit assignments

The SMMU_SCR0 bit assignments are:



a. Reserved in SMMUv1

Bits[31:30] Reserved.

NSCFG, bits[29:28]

Non-secure configuration.

This field only exists in SMMU_SCR0. In SMMU_CR0, these bits are reserved.

This field applies only to SSD Secure transactions bypassing the SMMU stream mapping process. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

`0b00` Use the default NS attribute

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Secure

`0b11` Non-secure.

These bits reset to 0.

WACFG, bits[27:26]

Write-Allocate configuration, controls the allocation hint for write accesses.

This field applies to transactions that bypass the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

`0b00` Default attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Write-Allocate.

`0b11` No Write-Allocate.

These bits reset to 0.

In an implementation that does not support Write-Allocate configuration, this field is RAZ/WI.

RACFG, bits[25:24]

Read-Allocate configuration. Controls the allocation hint for read accesses.

Applies to transactions that bypass the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

0b00 Default allocation attributes.

0b01 Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

0b10 Read-Allocate.

0b11 No Read-Allocate.

These bits reset to 0.

In an implementation that does not support Read-Allocate configuration, this field is RAZ/WI.

SHCFG, bits[23:22]

Shared configuration.

Applies to transactions that bypass the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

0b00 Default Shareable attribute.

0b01 Outer Shareable.

0b10 Inner Shareable.

0b11 Non-shareable.

These bits reset to 0.

In an implementation that does not support Shared configuration, this field is RAZ/WI.

SMCFCFG, bit[21]

Stream match conflict fault configuration. Controls transactions with multiple matches in the Stream mapping table.

The possible values of this bit are:

0 Permit the transaction to bypass the SMMU.

1 Raise a Stream match conflict fault.

It is IMPLEMENTATION DEFINED whether:

- The SMMU guarantees the detection of every Stream match conflict. See *StreamID matching on page 2-57* for more information.
- Stream match conflict handling is configurable. If not configurable, the value of SMCFCFG is fixed and writes are ignored.

This bit resets to an UNKNOWN value.

MTCFG, bit[20]

Memory Type Configuration, applies to transactions that bypass the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The possible values of this bit are:

0 Use the default memory attributes.

1 Use the MemAttr field for memory attributes.

This bit resets to 0.

MemAttr, bits[19:16]

Memory Attributes. See *Memory attribute, MemAttr on page 9-179*.

If SMMU_CR0.MTCFG==1, the memory attributes can be overlaid.

Note

When SMMU_CR0.MTCFG==1, restrictions apply when the memory is any type of Device memory, or is Normal Outer Non-cacheable or Inner Non-cacheable memory. See *SMMU_S2CRn, Stream-to-Context Register on page 9-220* for more information.

These bits reset to an UNKNOWN value.

BSU, bits[15:14]

Barrier Shareability Upgrade.

Upgrades the required shareability domain of barriers issued by client devices that are not mapped to a translation context bank, by setting the minimum shareability domain applied to any barrier.

Applies to transactions bypassing the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

`0b00` No effect.

`0b01` Inner Shareable.

`0b10` Outer Shareable.

`0b11` Full system.

This functionality might not be supported by all system topologies. In an implementation that does not support it, BSU is RAZ/SBZP.

These bits reset to an UNKNOWN value.

FB, bit[13] Force Broadcast of TLB, branch predictor, and instruction cache maintenance operations. Applies to transactions bypassing the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

Affects client TLB maintenance, BPIALL, and ICIALLU operations. If FB==1, any affected operation is modified to the equivalent broadcast variant in the Inner Shareable domain. The possible values of this bit are:

0 Process affected operations as presented.

1 Upgrade affected operations to be broadcast within the Inner Shareable domain.

This bit resets to an UNKNOWN value.

PTM, bit[12] Private TLB Maintenance. The possible values of this bit are:

0 The SMMU participates in broadcast TLB maintenance with the wider system, if supported in the implementation and as indicated by SMMU_IDR0.BTM.

1 SMMU TLBs are privately managed and are not required to respond to broadcast TLB maintenance operations from the wider system.

The PTM field is a hint. A broadcast TLB Invalidate operation is still permitted to affect all cached translations that are unlocked in the SMMU.

If SMMU_IDR0.BTM is zero, this field can be implemented as RAO/WI.

If SMMU_IDR0.BTM is one, an implementation that does not support private TLB maintenance can implement this field as RAZ/WI.

It is IMPLEMENTATION DEFINED whether Secure Hypervisor TLB entries are subject to broadcast TLB maintenance operations. It is also IMPLEMENTATION DEFINED whether SMMU_SCR0.PTM or SMMU_CR0.PTM controls whether they are affected by broadcast TLB maintenance. For this reason, ARM strongly recommends that software using Secure Hypervisor entries always maintains these TLB entries directly using the register interface.

This bit resets to an UNKNOWN value.

Bit[11] Reserved.

USFCFG, bit[10]

Unidentified stream fault configuration. The possible values of this bit are:

0 Permit any transaction that does not match any entries in the Stream mapping table to pass through.

1 Raise an Unidentified stream fault on any transaction that does not match any Stream mapping table entries.

This bit resets to an UNKNOWN value.

GSE, bit[9] Global Stall Enable. The possible values of this bit are:

0 Do not enforce global stalling across contexts.

1 Enforce global stalling across contexts.

This field is only writable if the implementation has global stall behavior. Otherwise, it is RAZ/WI.

This bit resets to 0.

STALLD, bit[8]

Stall Disable. The possible values of this bit are:

0 Permit per-context stalling on context faults.

1 Disable per-context stalling on context faults.

Setting this bit to 1 disables SMMU_SCR0.GSE and causes SMMU_SCR0.GSE to be RAZ/WI.

It is IMPLEMENTATION DEFINED whether the stall model is supported. When the stall model is not supported, this bit is UNKNOWN and might ignore writes.

In an implementation that supports two Security states, SMMU_CR0.STALLD must apply to a Non-secure translation context bank, and must affect SMMU_CR0.GSE. It is permitted to apply to a Secure translation context bank, and is permitted to affect SMMU_SCR0.GSE.

In an implementation that supports two Security states, this bit resets to 0. In an implementation that does not support two Security states, it resets to an UNKNOWN value.

TRANSIENTCFG, bits[7:6]

Transient Configuration, controls the transient allocation hint.

Applies to any transaction that bypasses the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

`0b00` Default transient allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Non-transient.

`0b11` Transient.

These bits reset to 0.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, these bits are RAZ/WI.

GCFGFIE, bit[5]

Global configuration fault interrupt enable. The possible values of this bit are:

0 Do not raise an interrupt on a Global configuration fault.

1 Raise an interrupt on a Global configuration fault.

This bit resets to 0.

This bit is RAZ/WI in implementations that do not support generation of configuration faults.

GCFGFRE, bit[4]

Global configuration fault report enable. The possible values of this bit are:

0 Do not return an abort on a Global configuration fault.

1 Return an abort on a Global configuration fault.

This bit resets to 0.

This bit is RAZ/WI in implementations that do not support generation of configuration faults.

EXIDENABLE, Bit[3]

Extended ID Enable. The possible values of this bit are:

0 Use the SMMUv1 format SMMU_SMRn.

1 Use the Extended ID format SMMU_SMRn.

In SMMUv1, this bit is reserved.

This bit resets to 0. This provides backwards compatibility with SMMUv1.

This feature is optional, and is supported when SMMU_IDR0.EXIDS==1. If not supported, this bit is RAZ/WI.

If stream matching is not supported, that is, when SMMU_IDR0.SMS==0, this bit is RAZ/WI.

Caution

- Changing the value of this bit under certain circumstances can result in UNPREDICTABLE behavior. ARM recommends changing the value of this bit only when, for all Stream Match Register groups of the appropriate Security state, both:
  - SMMU_S2CRn.EXIDVALID==0.
  - SMMU_SMRn.EXMASK[15]==0 or SMMU_SMRn.VALID==0, as appropriate.
- The reset values of SMMU_SMRn and SMMU_S2CRn are UNKNOWN, and therefore must be initialized before changing the value of this bit.

GFIE, bit[2] Global fault interrupt enable. The possible values of this bit are:

0 Do not raise an interrupt on a Global fault.

1 Raise an interrupt on a Global fault.

This bit resets to 0.

GFRE, bit[1] Global fault report enable. The possible values of this bit are:

0 Do not return an abort on a Global fault.

1 Return an abort on a Global fault.

This bit resets to 0.

For exclusive access transactions, when this bit is set to 0, ARM recommends that the SMMU reports the transaction as failed. See *Reporting exclusive access transactions on page 3-122* for more information.

CLIENTPD, bit[0]

Client Port Disable. The possible values of this bit are:

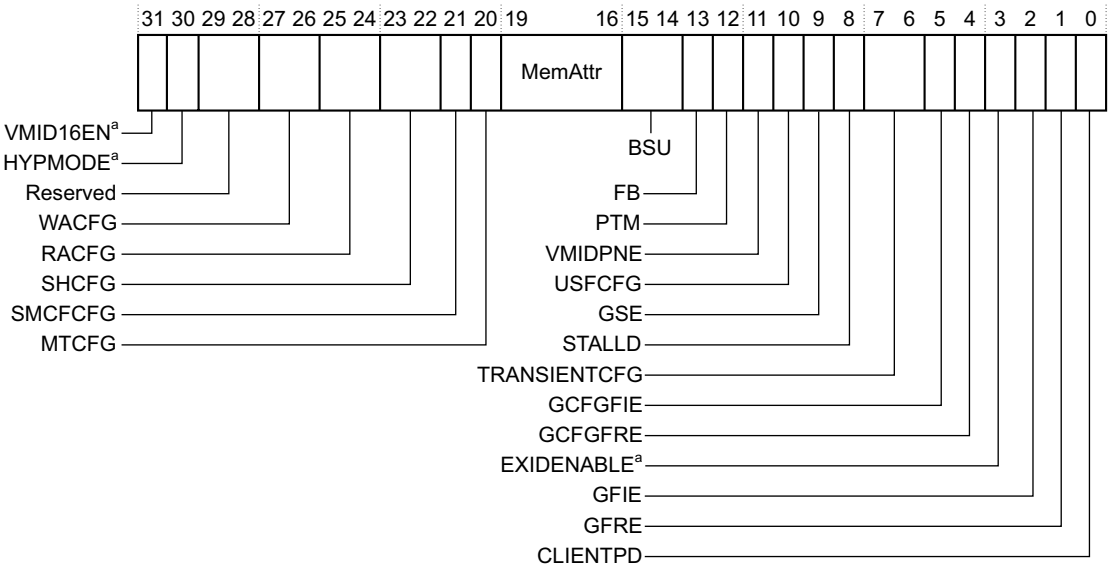
0 Each SMMU client access is subject to SMMU translation, access control, and attribute generation.

1 Each SMMU client access bypasses SMMU translation, access control, and attribute generation.

This bit resets to 1.

SMMU_CR0, Non-secure Configuration Register 0 bit assignments

The SMMU_CR0 bit assignments are:



a. Reserved in SMMUv1

VMID16EN, bit[31]

If 16-bit VMIDs are supported, as indicated by SMMU_IDR2.VMID16S, then enable 16-bit VMIDs. The possible values of this bit are:

0 VMIDs are 8 bits and held in SMMU_CBARn.VMID, SMMU_CBA2Rn.VMID16 is RES0.

1 VMIDs are 16 bits and held in SMMU_CBA2Rn.VMID16, SMMU_CBARn.VMID is RES0.

See *VMID size on page 1-39* for more information about the effect of changing the value of this bit.

In SMMUv1, this bit is reserved.

In SMMUv2, if 16-bit VMIDs are not supported, this bit is RES0.

HYPMODE, bit[30]

Hypervisor Mode. In an SMMUv2 implementation that supports E2HC, this bit selects which hypervisor context is used.

The possible values of this bit are:

0 When SMMU_CBARn.TYPE == 0b01, SMMU_CBARn[10] represents HYPC.

1 When SMMU_CBARn.TYPE == 0b01, SMMU_CBARn[10] represents E2HC.

If this bit is changed, TLB maintenance is required. See *E2H contexts (E2HC) on page 2-90* for more information about this bit.

This bit is RES0 in SMMUv1 and in an SMMUv2 implementation that does not support E2HC. Otherwise, this bit resets to 0.

Bits[29:28] Reserved.

WACFG, bits[27:26]

Write-Allocate configuration, controls the allocation hint for write accesses.

This field applies to transactions that bypass the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

`0b00` Default attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Write-Allocate.

`0b11` No Write-Allocate.

These bits reset to 0.

In an implementation that does not support Write-Allocate configuration, this field is RAZ/WI.

RACFG, bits[25:24]

Read-Allocate configuration. Controls the allocation hint for read accesses.

Applies to transactions that bypass the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

`0b00` Default allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Read-Allocate.

`0b11` No Read-Allocate.

These bits reset to 0.

In an implementation that does not support Read-Allocate configuration, this field is RAZ/WI.

SHCFG, bits[23:22]

Shared configuration.

Applies to transactions that bypass the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

`0b00` Default Shareable attribute.

`0b01` Outer Shareable.

`0b10` Inner Shareable.

`0b11` Non-shareable.

These bits reset to 0.

In an implementation that does not support Shared configuration, this field is RAZ/WI.

SMCFCFG, bit[21]

Stream match conflict fault configuration. Controls transactions with multiple matches in the Stream mapping table.

The possible values of this bit are:

`0` Permit the transaction to bypass the SMMU.

`1` Raise a Stream match conflict fault.

It is IMPLEMENTATION DEFINED whether:

- The SMMU guarantees the detection of every Stream match conflict. See *StreamID matching on page 2-57* for more information.
- Stream match conflict handling is configurable. If not configurable, the value of SMCFCFG is fixed and writes are ignored.

This bit resets to an UNKNOWN value.

MTCFG, bit[20]

Memory Type Configuration, applies to transactions that bypass the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The possible values of this bit are:

`0` Use the default memory attributes.

`1` Use the MemAttr field for memory attributes.

This bit resets to 0.

MemAttr, bits[19:16]

Memory Attributes. See *Memory attribute, MemAttr on page 9-179*.

If SMMU_CR0.MTCFG==1, the memory attributes can be overlaid.

Note

When SMMU_CR0.MTCFG==1, restrictions apply when the memory is any type of Device memory, or is Normal Outer Non-cacheable or Inner Non-cacheable memory. See *SMMU_S2CRn, Stream-to-Context Register on page 9-220* for more information.

These bits reset to an UNKNOWN value.

BSU, bits[15:14]

Barrier Shareability Upgrade.

Upgrades the required shareability domain of barriers issued by client devices that are not mapped to a translation context bank, by setting the minimum shareability domain applied to any barrier.

Applies to transactions bypassing the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

`0b00` No effect.

`0b01` Inner Shareable.

`0b10` Outer Shareable.

`0b11` Full system.

This functionality might not be supported by all system topologies. In an implementation that does not support it, BSU is RAZ/SBZP.

These bits reset to an UNKNOWN value.

FB, bit[13] Force Broadcast of TLB, branch predictor, and instruction cache maintenance operations. Applies to transactions bypassing the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

Affects client TLB maintenance, BPIALL, and ICIALLU operations. If FB==1, any affected operation is modified to the equivalent broadcast variant in the Inner Shareable domain. The possible values of this bit are:

0 Process affected operations as presented.

1 Upgrade affected operations to be broadcast within the Inner Shareable domain.

This bit resets to an UNKNOWN value.

PTM, bit[12] Private TLB Maintenance. The possible values of this bit are:

0 The SMMU participates in broadcast TLB maintenance with the wider system, if supported in the implementation and as indicated by SMMU_IDR0.BTM.

1 SMMU TLBs are privately managed and are not required to respond to broadcast TLB maintenance operations from the wider system.

The PTM field is a hint. A broadcast TLB Invalidate operation is still permitted to affect all cached translations that are unlocked in the SMMU.

If SMMU_IDR0.BTM is zero, this field can be implemented as RAO/WI.

If SMMU_IDR0.BTM is one, an implementation that does not support private TLB maintenance can implement this field as RAZ/WI.

It is IMPLEMENTATION DEFINED whether Secure Hypervisor TLB entries are subject to broadcast TLB maintenance operations. It is also IMPLEMENTATION DEFINED whether SMMU_SCR0.PTM or SMMU_CR0.PTM controls whether they are affected by broadcast TLB maintenance. For this reason, ARM strongly recommends that software using Secure Hypervisor entries always maintains these TLB entries directly using the register interface.

This bit resets to an UNKNOWN value.

VMIDPNE, bit[11]

VMID Private Namespace Enable. The possible values of this bit are:

0 SMMU values are coordinated with the wider system.

1 SMMU VMID values are a private namespace, not coordinated with the wider system.

If VMIDPNE==1, broadcast TLB Invalidate operations specifying a VMID value are not required to apply to cached translations in the SMMU.

The VMIDPNE field is a hint. A broadcast TLB Invalidate operation is still permitted to affect all cached translations that are unlocked in the SMMU.

If SMMU_IDR0.BTM is zero, this field might be RAO/WI.

If SMMU_IDR0.BTM is one, an implementation that does not support a private namespace can implement this field as RAZ/WI.

This bit resets to an UNKNOWN value.

In SMMU_SCR0, VMIDPNE is reserved.

USFCFG, bit[10]

Unidentified stream fault configuration. The possible values of this bit are:

0 Permit any transaction that does not match any entries in the Stream mapping table to pass through.

1 Raise an Unidentified stream fault on any transaction that does not match any Stream mapping table entries.

This bit resets to an UNKNOWN value.

GSE, bit[9] Global Stall Enable. The possible values of this bit are:

0 Do not enforce global stalling across contexts.

1 Enforce global stalling across contexts.

This field is only writable if the implementation has global stall behavior. Otherwise, it is RAZ/WI.

This bit resets to 0.

STALLD, bit[8]

Stall Disable. The possible values of this bit are:

0 Permit per-context stalling on context faults.

1 Disable per-context stalling on context faults.

Setting this bit to 1 disables SMMU_CR0.GSE and causes SMMU_CR0.GSE to be RAZ/WI.

It is IMPLEMENTATION DEFINED whether the stall model is supported. When the stall model is not supported, this bit is UNKNOWN and might ignore writes.

In an implementation that supports two Security states, SMMU_CR0.STALLD must apply to a Non-secure translation context bank, and must affect SMMU_CR0.GSE. It is permitted to apply to a Secure translation context bank, and is permitted to affect SMMU_SCR0.GSE.

In an implementation that supports two Security states, this bit resets to 0. In an implementation that does not support two Security states, it resets to an UNKNOWN value.

TRANSIENTCFG, bits[7:6]

Transient Configuration, controls the transient allocation hint.

Applies to any transaction that bypasses the Stream mapping table. See *Bypassing the Stream mapping table on page 2-56*.

The encoding of this field is:

`0b00` Default transient allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Non-transient.

`0b11` Transient.

These bits reset to 0.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, these bits are RAZ/WI.

GCFGFIE, bit[5]

Global configuration fault interrupt enable. The possible values of this bit are:

0 Do not raise an interrupt on a Global configuration fault.

1 Raise an interrupt on a Global configuration fault.

This bit resets to 0.

This bit is RAZ/WI in implementations that do not support generation of configuration faults.

GCFGFRE, bit[4]

Global configuration fault report enable. The possible values of this bit are:

0 Do not return an abort on a Global configuration fault.

1 Return an abort on a Global configuration fault.

This bit resets to 0.

This bit is RAZ/WI in implementations that do not support generation of configuration faults.

EXIDENABLE, Bit[3]

Extended ID Enable. The possible values of this bit are:

0 Use the SMMUv1 format SMMU_SMRn.

1 Use the Extended ID format SMMU_SMRn.

In SMMUv1, this bit is reserved.

This bit resets to 0. This provides backwards compatibility with SMMUv1.

This feature is optional, and is supported when SMMU_IDR0.EXIDS==1. If not supported, this bit is RAZ/WI.

If stream matching is not supported, that is, when SMMU_IDR0.SMS==0, this bit is RAZ/WI.

Caution

- Changing the value of this bit under certain circumstances can result in UNPREDICTABLE behavior. ARM recommends changing the value of this bit only when, for all Stream Match Register groups of the appropriate Security state, both:
  - SMMU_S2CRn.EXIDVALID==0.
  - SMMU_SMRn.EXMASK[15]==0 or SMMU_SMRn.VALID==0, as appropriate.
- The reset values of SMMU_SMRn and SMMU_S2CRn are UNKNOWN, and therefore must be initialized before changing the value of this bit.

GFIE, bit[2] Global fault interrupt enable. The possible values of this bit are:

0 Do not raise an interrupt on a Global fault.

1 Raise an interrupt on a Global fault.

This bit resets to 0.

GFRE, bit[1] Global fault report enable. The possible values of this bit are:

0 Do not return an abort on a Global fault.

1 Return an abort on a Global fault.

This bit resets to 0.

For exclusive access transactions, when this bit is set to 0, ARM recommends that the SMMU reports the transaction as failed. See *Reporting exclusive access transactions on page 3-122* for more information.

CLIENTPD, bit[0]

Client Port Disable. The possible values of this bit are:

0 Each SMMU client access is subject to SMMU translation, access control, and attribute generation.

1 Each SMMU client access bypasses SMMU translation, access control, and attribute generation.

This bit resets to 1.

9.6.4 SMMU_*s*CR2, Configuration Register 2

The SMMU_*s*CR2 characteristics are:

Purpose Provides top-level control of the SMMU.

Usage constraints There are no usage constraints.

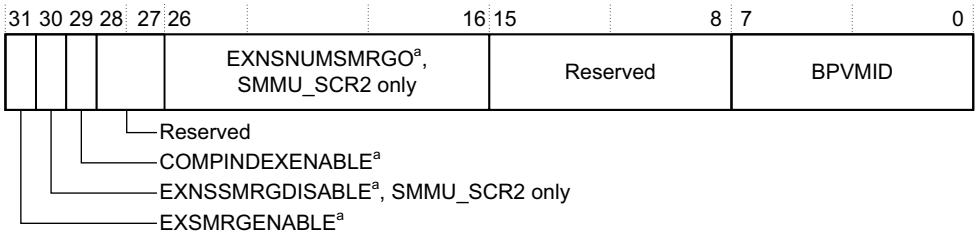
Configurations In an implementation that supports two Security states, this register is Banked.

SMMU_NSCR2 is provided as a Secure alias of the Non-secure SMMU_CR2. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

For more information about the Extended Stream Matching Extension, see Chapter 14 *Extended Stream Matching Extension*.

Attributes A 32-bit RW register. See Table 9-1 on page 9-170.

The SMMU_*s*CR2 bit assignments:



a. RES0 in SMMUv1

EXSMRGENABLE, bit[31]

In SMMUv2 when the Extended Stream Matching Extension is implemented, enables the use of that extension. The extension provides the Extended Stream Match Register groups.

The possible values of this bit are:

0 Use of Extended Stream Match Register groups disabled.

1 Use of Extended Stream Match Register groups enabled.

If the value of SMMU_SCR2.EXNSSMRGDISABLE is 1, then SMMU_CR2.EXSMRGENABLE is treated as 0 for all purposes other than reading back the value written to the bit.

In SMMUv1, or if the Extended Stream Matching Extension is not implemented, this bit is RES0.

When implemented as an R/W bit, this bit resets to 0.

EXNSSMRGDISABLE, bit[30], SMMU_SCR2 only

In SMMUv2 when the Extended Stream Matching Extension is implemented, disables the use of Extended Stream Match Register Groups by Non-secure software.

The possible values for this bit are:

0 Use of Extended Stream Match Register Groups by Non-secure software is enabled.

1 Use of Extended Stream Match Register Groups by Non-secure software is disabled.

In SMMUv1, or if the Extended Stream Matching Extension is not implemented, this bit is RES0.

When implemented as an R/W bit, this bit resets to 0.

Bit[30], SMMU_CR2 only

RES0.

COMPINDEXENABLE, bit[29]

StreamID compressed indexed matching enable. The possible values of this bit are:

0 Compressed indexing mechanism is disabled for the Security state. This means that normal stream matching occurs.

1 Compressed indexing mechanism is enabled for the Security state, and the SMRs are ignored.

The Non-secure copy of SMMU_CR2.COMPINDEXENABLE is RAZ/WI if SMMU_SCR1.NSCOMPINDEXDISABLE is 1.

The Secure copy of SMMU_sCR2.COMPINDEXENABLE is RAZ/WI if SMMU_SCR1.NSCOMPINDEXDISABLE is 0.

In SMMUv1, or if the StreamID Compressed Indexing Extension is not implemented, this bit is RES0.

See Chapter 15 *StreamID Compressed Indexing Extension* for more information.

Bits[28:27] Reserved

EXNSNUMSMRGO, bits[26:16], SMMU_SCR2 only

In an SMMUv2 implementation that includes the Extended Stream Matching extension, when that extension is enabled, defines the number of Extended Stream Match Register groups that are reserved for Non-secure contexts. These Non-secure groups start at the first implemented Extended Stream Match register group. For more information, see *The number of Extended Stream Match Register groups available to Non-secure transactions on page 14-277*.

When the value of SMMU_SCR2.EXNSSMRGDISABLE is 1, this field is treated as zero for all purposes other than reading back the register value.

When the value of SMMU_SCR2.EXSMRGENABLE is 0, the value of this field is ignored for all purposes other than reading back the register value.

In SMMUv1, or if the Extended Stream Matching Extension is not implemented, this field is RES0.

Bits[26:16], SMMU_CR2 only

RES0.

Bits[15:8] Reserved

BPVMID, bits[7:0]

Bypass VMID.

This field is reserved for IMPLEMENTATION DEFINED use as a VMID field applied to client transactions that bypass the SMMU. See *Bypassing the Stream mapping table on page 2-56*.

Whether this field is implemented, and the effect it has, is IMPLEMENTATION DEFINED.

In an implementation that does not support this field, these bits are RAZ/WI.

If an implementation that supports two Security states does not provide a VMID for Secure translation regimes, any Secure VMID field is ignored. See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

Note

This field remains as 8 bits even if 16-bit VMIDs are supported when using MMU translations, that is, SMMU_IDR2.VMID16S is 1.

9.6.5 SMMU_*s*GATS1PR, GAT Stage 1 Privileged Read

The SMMU_sGATS1PR characteristics are:

Purpose Translates an argument-supplied address. Writes the result to SMMU_sGPAR.

Usage constraints The Security state of the transaction determines whether SMMU_GATS1PR or SMMU_SGATS1PR is used.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations If the implementation supports two Security states, this register is in SMMU_SGATS1PR. Otherwise, it is in SMMU_GATS1PR.

SMMU_NSGATS1PR is provided as a Secure alias of the Non-secure SMMU_GATS1PR. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

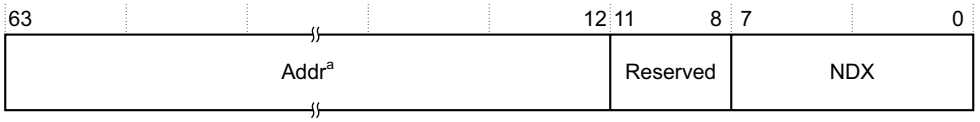
In SMMUv2, all address translation registers are OPTIONAL.

Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_*s*GATS1PR bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

The translation is performed as though the address is associated with a privileged read.

In SMMUv1, Addr is bits[31:12].

Bits[11:8] Reserved.

NDX, bits[7:0] Translation context bank index for stage 1 translation.

See *Address translation registers in the global address space on page 4-130*.

9.6.6 SMMU_*s*GATS1PW, GAT Stage 1 Privileged Write

The SMMU_sGATS1PW characteristics are:

Purpose Translates an argument-supplied input address. Writes the result to SMMU_sGPAR.

Usage constraints The Security state of the transaction determines whether SMMU_GATS1PW or SMMU_SGATS1PW is used.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations If the implementation supports two Security states, this register is in SMMU_SGATS1PW. Otherwise, it is in SMMU_GATS1PW.

SMMU_NSGATS1PW is provided as a Secure alias of the Non-secure SMMU_GATS1PW. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

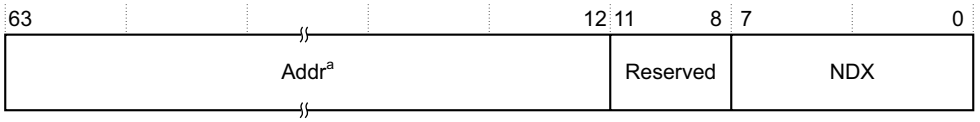
In SMMUv2, all address translation registers are OPTIONAL.

Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_*s*GATS1PW bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

The translation is performed as though the address is associated with a privileged write.

In SMMUv1, Addr is bits[31:12].

Bits[11:8] Reserved.

NDX, bits[7:0] Translation context bank index for stage 1 translation.

See *Address translation registers in the global address space on page 4-130*.

9.6.7 SMMU_*s*GATS1UR, GAT Stage 1 Unprivileged Read

The SMMU_sGATS1UR characteristics are:

Purpose Translates an argument-supplied input address. Writes the result to SMMU_sGPAR.

Usage constraints The Security state of the transaction determines whether SMMU_GATS1UR or SMMU_SGATS1UR is used.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations If the implementation supports two Security states, this register is available in SMMU_SGATS1UR. Otherwise, it is in SMMU_GATS1UR.

SMMU_NSGATS1UR is provided as a Secure alias of the Non-secure SMMU_GATS1UR. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

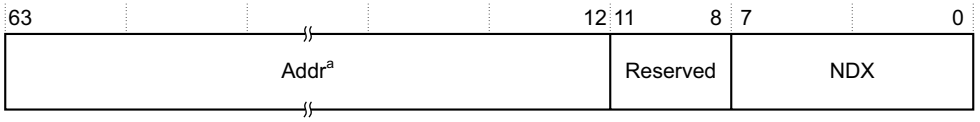
In SMMUv2, all address translation registers are OPTIONAL.

Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_*s*GATS1UR bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

Performed as though the address is associated with an unprivileged read.

In SMMUv1, Addr is bits[31:12].

Bits[11:8] Reserved.

NDX, bits[7:0] Translation context bank index for stage 1 translation.

See *Address translation registers in the global address space on page 4-130*.

9.6.8 SMMU_*s*GATS1UW, GAT Stage 1 Unprivileged Write

The SMMU_sGATS1UW characteristics are:

Purpose Translates an argument-supplied input address. Writes the result to SMMU_sGPAR.

Usage constraints The Security state of the transaction determines whether SMMU_GATS1UW or SMMU_SGATS1UW is used.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations If the implementation supports two Security states, this register is in SMMU_SGATS1UW. Otherwise, it is in SMMU_GATS1UW.

SMMU_NSGATS1UW is provided as a Secure alias of the Non-secure SMMU_GATS1UW. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

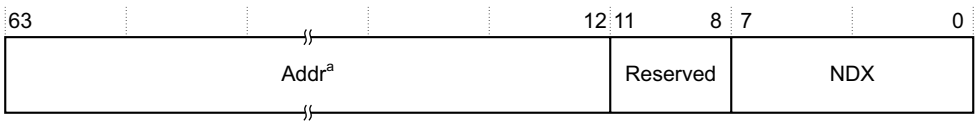
In SMMUv2, all address translation registers are OPTIONAL.

Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_*s*GATS1UW bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

The translation is performed as though the address is associated with an unprivileged write.

In SMMUv1, Addr is bits[31:12].

Bits[11:8] Reserved.

NDX, bits[7:0] Translation context bank index for stage 1 translation.

See *Address translation registers in the global address space on page 4-130*.

9.6.9 SMMU_*s*GATS12PR, GAT Stages 1 and 2 Privileged Read

The SMMU_sGATS12PR characteristics are:

Purpose Translates an argument-supplied input address. Writes the result to SMMU_sGPAR.

Usage constraints The Security state of the transaction determines whether SMMU_GATS12PR or SMMU_SGATS12PR is used.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations If the implementation supports two Security states, this register is available in SMMU_SGATS12PR. Otherwise, it is in SMMU_GATS12PR.

SMMU_NSGATS12PR is provided as a Secure alias of the Non-secure SMMU_GATS12PR. See Table 9-1 on page 9-170. See also *Secure alias for Non-secure registers on page 9-177* for general information.

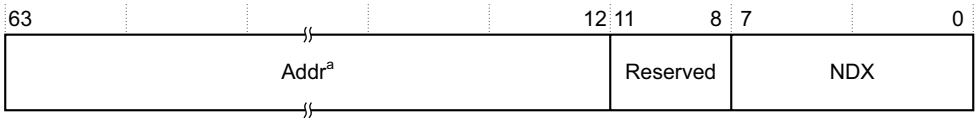
In SMMUv2, all address translation registers are OPTIONAL.

Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_*s*GATS12PR bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

The translation is performed as though the address is associated with a privileged read.

Bits[11:8] Reserved.

NDX, bits[7:0] Translation context bank index for stage 1 translation.

See *Address translation registers in the global address space on page 4-130*.

9.6.10 SMMU_*s*GATS12PW, GAT Stages 1 and 2 Privileged Write

The SMMU_sGATS12PW characteristics are:

Purpose Translates an argument-supplied input address. Writes the result to SMMU_sGPAR.

Usage constraints The Security state of the transaction determines whether SMMU_GATS12PW or SMMU_SGATS12PW is used.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations If the implementation supports two Security states, this register is in SMMU_SGATS12PW. Otherwise, it is in SMMU_GATS12PW.

SMMU_NSGATS12PW is provided as a Secure alias of the Non-secure SMMU_GATS12PW. See Table 9-1 on page 9-170. See also *Secure alias for Non-secure registers on page 9-177* for general information.

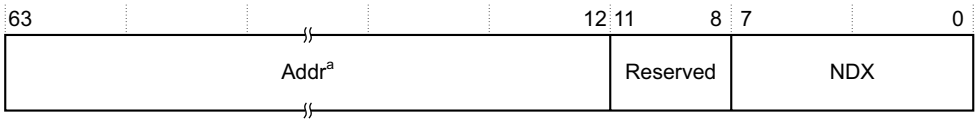
In SMMUv2, all address translation registers are OPTIONAL.

Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_*s*GATS12PW bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

The translation is performed as though the address is associated with a privileged write.

Bits[11:8] Reserved.

NDX, bits[7:0] Translation context bank index for stage 1 translation.

See *Address translation registers in the global address space on page 4-130*.

9.6.11 SMMU_*s*GATS12UR, GAT Stages 1 and 2 Unprivileged Read

The SMMU_sGATS12UR characteristics are:

Purpose Translates an argument-supplied input address. Writes the result to SMMU_sGPAR.

Usage constraints The Security state of the transaction determines whether SMMU_GATS12UR or SMMU_SGATS12UR is used.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations If the implementation supports two Security states, this register is in SMMU_SGATS12UR. Otherwise, it is in SMMU_GATS12UR.

SMMU_NSGATS12UR is provided as a Secure alias of the Non-secure SMMU_GATS12UR. See Table 9-1 on page 9-170. See also *Secure alias for Non-secure registers on page 9-177* for general information.

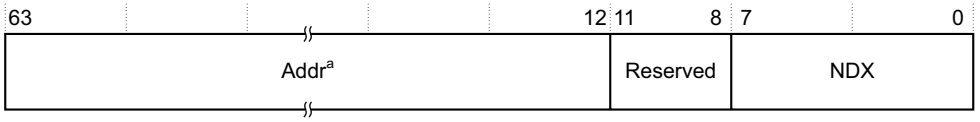
In SMMUv2, all address translation registers are OPTIONAL.

Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_*s*GATS12UR bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

The translation is performed as though the address is associated with an unprivileged read.

Bits[11:8] Reserved.

NDX, bits[7:0]

Translation context bank index for stage 1 translation.

See *Address translation registers in the global address space on page 4-130*.

9.6.12 SMMU_*s*GATS12UW, GAT Stages 1 and 2 Unprivileged Write

The SMMU_sGATS12UW characteristics are:

Purpose Translates an argument-supplied input address. Writes the result to SMMU_sGPAR.

Usage constraints The Security state of the transaction determines whether SMMU_GATS12UW or SMMU_SGATS12UW is used.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations If the implementation supports two Security states, this register is in SMMU_SGATS12UW. Otherwise, it is in SMMU_GATS12UW.

SMMU_NSGATS12UW is provided as a Secure alias of the Non-secure SMMU_GATS12UW. See Table 9-1 on page 9-170. See also *Secure alias for Non-secure registers on page 9-177* for general information.

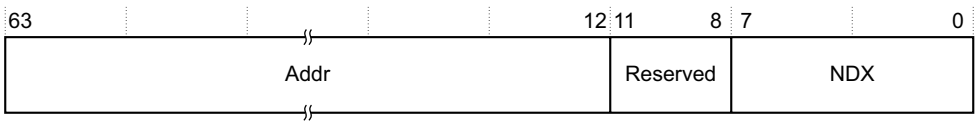
In SMMUv2, all address translation registers are OPTIONAL.

Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_*s*GATS12UW bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

The translation is performed as though the address is associated with an unprivileged write.

Bits[11:8] Reserved.

NDX, bits[7:0]

Translation context bank index for stage 1 translation.

See *Address translation registers in the global address space on page 4-130*.

9.6.13 SMMU_sGATSR, Global Address Translation Status Register

The SMMU_*s*GATSR characteristics are:

Purpose Gives status information pertaining to active global address translation operations.

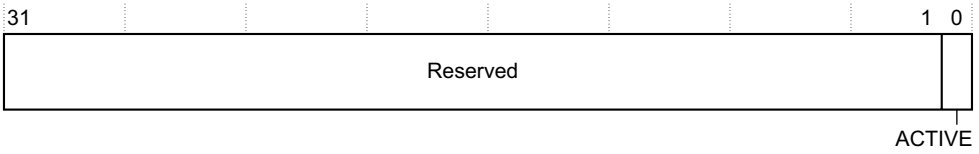
Usage constraints If software requests a new address translation operation while an existing operation is active, the result is UNPREDICTABLE. See *Usage model on page 4-127* for more information.

Configurations If the implementation supports two Security states, this register is Banked.

SMMU_NSGATSR is provided as a Secure alias of the Non-secure SMMU_GATSR. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

Attributes A 32-bit RO register with an UNKNOWN reset value. See also Table 9-1 on page 9-170.

The SMMU_*s*GATSR bit assignments are:



Bits[31:1] Reserved.

ACTIVE, bit[0]

Address Translation Active.

The possible values of this bit are:

0 Address translation not active.

1 Address translation active, SMMU_sGPAR is yet to be updated.

9.6.14 SMMU_sGFAR, Global Fault Address Register

The SMMU_*s*GFAR characteristics are:

Purpose Contains the input address of an erroneous request reported by SMMU_sGFSR.

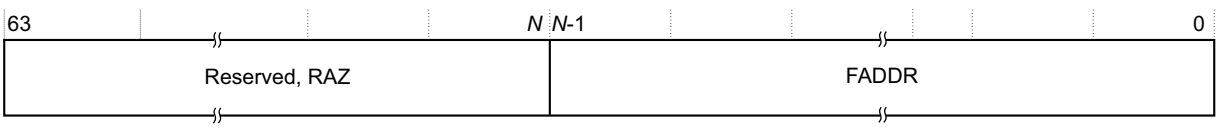
Usage constraints See *Handling multiple memory faults on page 3-96* for information about when this register is updated. See also *Multiple faults on page 3-106*.

Configurations In an implementation that supports two Security states, this register is Banked.

SMMU_NSGFAR is provided as a Secure alias of the Non-secure SMMU_GFAR. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

Attributes A 64-bit RW register with an UNKNOWN reset value. See also Table 9-1 on page 9-170.

The SMMU_*s*GFAR bit assignments are:



Bits[63:*N*] Reserved.

FADDR, bits[*N*-1:0]

Fault address, the input address of the faulting access. For a configuration access fault, this is the offset from SMMU_BASE. For other fault classes, it is the input address of the faulting access, that the system can interpret in a number of ways.

In SMMUv2, the value of *N* depends on the implemented upstream bus size, as defined by SMMU_IDR2.UBS.

In SMMUv1, the value of *N* depends on the implemented input address space, as defined by SMMU_IDR2.IAS, and can be:

- 32, for 4GB of address space.
- 36, for 64GB of address space.
- 40, for 1TB of address space.

SMMU_IDR2.UBS gives the implemented upstream bus size.

9.6.15 SMMU_*s*GFSR, Global Fault Status Register

The SMMU_*s*GFSR characteristics are:

Purpose Gives the fault status for each of the following possible faults:

- Invalid context fault.
- Unidentified stream fault.
- Stream match conflict fault.
- Unimplemented context bank fault.
- Unimplemented context interrupt fault.
- Configuration access fault.
- External fault.

Usage constraints There are no usage constraints.

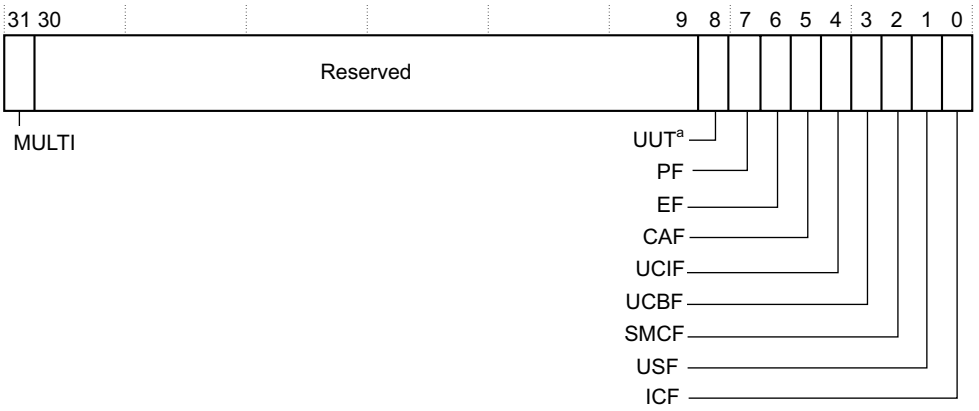
Configurations In an implementation that supports two Security states, this register is Banked.

SMMU_NSGFSR is provided as a Secure alias of the Non-secure SMMU_GFSR. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

Attributes A 32-bit RW clear register. A value of 1 written to any non-reserved bit clears that bit. A value of 0 written to any of these bits leaves the bit unchanged.

This register resets to an UNKNOWN value. See also Table 9-1 on page 9-170.

The SMMU_*s*GFSR bit assignments are:



a. Reserved in SMMUv1

MULTI, bit[31]

Multiple error condition. The possible values of this bit are:

0 No multiple error condition was encountered.

1 An error occurred while the value in SMMU_*s*GFSR was nonzero.

Bits[30:9] Reserved.

UUT, bit 8 Unsupported Upstream Transaction. The possible values of this bit are:

0 No unsupported upstream transaction fault

1 Unsupported upstream transaction fault caused by receipt of an unsupported client transaction from an upstream device.

In SMMUv1, this bit is reserved.

PF, bit[7] Permission fault.

In SMMU_GFSR, this field is reserved.

In SMMU_SGFSR, this field records global SMMU_SCR1.SIF faults.

The possible values of this bit are:

0 No permission fault.

1 Permission fault.

Note

If a transaction is associated with a particular translation context bank, faults are recorded in SMMU_CBn_FSR instead of SMMU_SGFSR.

EF, bit[6] External fault. The possible values of this bit are:

0 No external fault.

1 External fault caused by an external abort.

Note

This bit might be RAZ/WI.

CAF, bit[5] Configuration access fault. The possible values of this bit are:

0 No configuration access fault.

1 Configuration access fault.

Note

In implementations that do not support configuration access faults, this bit is RAZ/WI.

UCIF, bit[4] Unimplemented context interrupt fault. The possible values of this bit are:

0 No Unimplemented context interrupt fault.

1 Unimplemented context interrupt fault.

UCBF, bit[3] Unimplemented context bank fault. The possible values of this bit are:

0 No Unimplemented context bank fault.

1 Unimplemented context bank fault.

SMCF, bit[2] Stream match conflict fault. The possible values of this bit are:

0 No Stream match conflict fault.

1 Stream match conflict fault.

USF, bit[1] Unidentified stream fault. The possible values of this bit are:

0 No Unidentified stream fault.

1 Unidentified stream fault.

ICF, bit[0] Invalid context fault. The possible values of this bit are:

0 Invalid context fault.

1 No invalid context fault.

9.6.16 SMMU_*s*GFSRRESTORE, Global Fault Status Restore Register

The SMMU_*s*GFSRRESTORE characteristics are:

Purpose Restores the state of SMMU_sGFSR, after a reset, for example.

Usage constraints The SMMU_sGFSR register restored depends on the SMMU_*s*GFSRRESTORE register written to:

- Writing to SMMU_GFSRRESTORE restores SMMU_GFSR.
- Writing to SMMU_SGFSRRESTORE restores SMMU_SGFSR.

Configurations If the implementation supports two Security states, this register is Banked.

SMMU_NSGFSRRESTORE is provided as a Secure alias of the Non-secure SMMU_GFSRRESTORE. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

Attributes A 32-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_*s*GFSRRESTORE bit assignments are identical to SMMU_sGFSR. The value of this register overwrites the value of SMMU_sGFSR.

9.6.17 SMMU_*s*GFSYNR0, Global Fault Syndrome Register 0

The SMMU_*s*GFSYNR0 characteristics are:

Purpose Contains fault syndrome information relating to SMMU_sGFSR.

Usage constraints SMMU_GFSYNR0 is for Non-secure accesses. In an implementation that supports two Security states, SMMU_SGFSYNR0 is for Secure accesses.

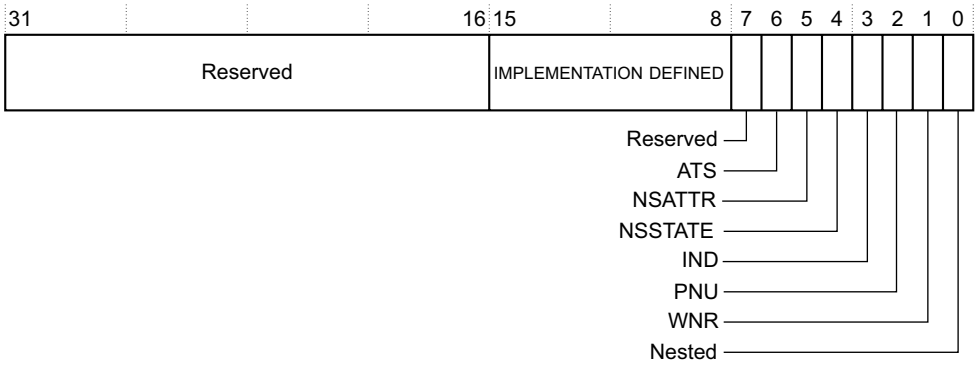
See *Multiple faults on page 3-106* for information about when this register is updated. See also *Multiple fault conditions on page 3-115*.

Configurations In an implementation that supports two Security states, this register is Banked.

SMMU_NSGFSYNR0 is provided as a Secure alias of the Non-secure SMMU_GFSYNR0. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

Attributes A 32-bit RW register that resets to an UNKNOWN value. See also Table 9-1 on page 9-170.

The SMMU_*s*GFSYNR0 bit assignments are:



Bits[31:16] Reserved.

Bits[15:8] IMPLEMENTATION DEFINED.

Bit[7] Reserved.

ATS, Bit[6] Address translation operation fault. The possible values of this bit are:

0 The fault was not caused by the processing of an SMMU_CBn_ATS operation initiated in a stage 1 translation context bank.

1 The fault was caused by the processing of an SMMU_CBn_ATS operation initiated in a stage 1 translation context bank. See *Fault handling within stage 1 followed by stage 2 translations initiated in a context bank on page 4-127* for more information.

NSATTR, bit[5]

0 The faulty transaction has the Secure attribute.

1 The faulty transaction has the Non-secure attribute.

In SMMU_GFSYNR0, this field is reserved.

NSSTATE, bit[4]

0 The faulty transaction is associated with a Secure device.

1 The faulty transaction is associated with a Non-secure device.

In SMMU_GFSYNR0, this field is reserved.

This field is set to 1 if a fault encountered when processing a Non-secure client transaction is reported to SMMU_SGFSR, for example, when:

- A configuration access fault occurs as a response to a Non-secure attempt to access a Secure-only resource, such as SMMU_SCR1, or a context bank allocated for use by Secure software.
- An external abort associated with Non-secure execution occurs, when the SMMU_SCR1.GEFRO bit is set to 1.

IND, bit[3] Instruction Not Data. The possible values of this bit are:

0 The faulty transaction has the data access attribute.

1 The faulty transaction has the instruction access attribute.

PNU, bit[2] Privileged Not Unprivileged. The possible values of this bit are:

0 The faulty transaction has the unprivileged attribute.

1 The faulty transaction is privileged.

WNR, bit[1] Write Not Read. The possible values of this bit are:

0 The faulty transaction is a read.

1 The faulty transaction is a write.

Note

For far atomics, this field will always indicate “Read”.

Nested, bit[0] Nested fault. The possible values of this bit are:

0 The fault occurred in the initial stream context.

1 The fault occurred in the stage 2 context associated with a stage 1 followed by stage 2 translation.

In SMMU_SGFSYNR0, this field is reserved.

9.6.18 SMMU_*s*GFSYNR1, Global Fault Syndrome Register 1

The SMMU_*s*GFSYNR1 characteristics are:

Purpose Contains fault syndrome information relating to SMMU_sGFSR.

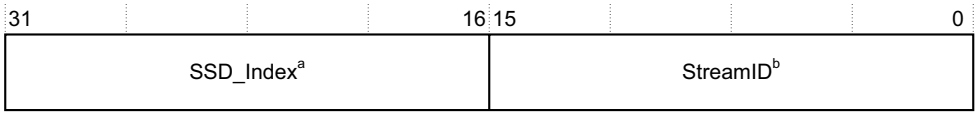
Usage constraints The Security state of the faulty transaction determines whether SMMU_GFSYNR1 or SMMU_SGFSYNR1 is used.

Configurations In an implementation that supports two Security states, this register is Banked.

SMMU_NSGFSYNR1 is provided as a Secure alias of the Non-secure SMMU_GFSYNR1. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

Attributes A 32-bit RW register that resets to an UNKNOWN value. See also Table 9-1 on page 9-170.

The SMMU_*s*GFSYNR1 bit assignments are:



a. In SMMUv1, bit[31] is reserved and bits[30:16] are SSD_Index.
b. In SMMUv1, bit[15] is reserved and bits[14:0] are StreamID.

SSD_Index, bits[31:16]

SSD_Index of the transaction that caused the fault.

The number of SSD_Index bits is IMPLEMENTATION DEFINED. Unimplemented bits behave as RAZ/WI.

The SSD_Index field is only accessible to configuration accesses by Secure software, using SMMU_NSGFSYNR1. Non-secure configuration accesses treat this field as RAZ/WI. This means that software must access SMMU_NSGFSYNR1 to obtain the SSD_Index for a faulty Non-secure transaction.

Note

When SMMU_SSDR registers are not provided in the SMMU address space, the SSD_Index field is UNKNOWN. See *TLB operation on page 2-72* for more information about the SMMU_SSDR registers.

SMMUv1 does not support 16-bit StreamIDs, and bit[31] is reserved.

StreamID, bits[15:0]

StreamID of the transaction that caused the fault.

The number of StreamID bits is IMPLEMENTATION DEFINED. Unimplemented bits behave as RAZ/WI.

SMMUv1 does not support 16-bit StreamIDs, and bit[15] is reserved.

9.6.19 SMMU_sGFSYNR2, Global Fault Syndrome Register 2

The SMMU_*s*GFSYNR2 characteristics are:

Purpose Contains fault syndrome information relating to faults in SMMU_sGFSR.

Usage constraints Any usage constraints are IMPLEMENTATION DEFINED.

Configurations In an implementation that supports two Security states, this register is Banked.

SMMU_NSGFSYNR2 is provided as a Secure alias of the Non-secure SMMU_GFSYNR2. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

Attributes A 32-bit RW register that resets to an UNKNOWN value. See also Table 9-1 on page 9-170.

The SMMU_*s*GFSYNR2 bit assignments are IMPLEMENTATION DEFINED.

9.6.20 SMMU_sGPAR, Global Physical Address Register

The SMMU_*s*GPAR characteristics are:

Purpose Holds the result of a global address translation operation.

Usage constraints The Security state of the transaction determines whether SMMU_GPAR or SMMU_SGPAR is used.

Configurations The format of this register depends on the translation scheme selected, and whether the last global address translation operation was successful. See:

- *AArch32 Short-descriptor translation scheme*.
- *AArch32 Long-descriptor and AArch64 translation schemes on page 9-216*.
- *Fault format on page 9-217*.

See also *Multi-format registers and reserved fields on page 9-180*.

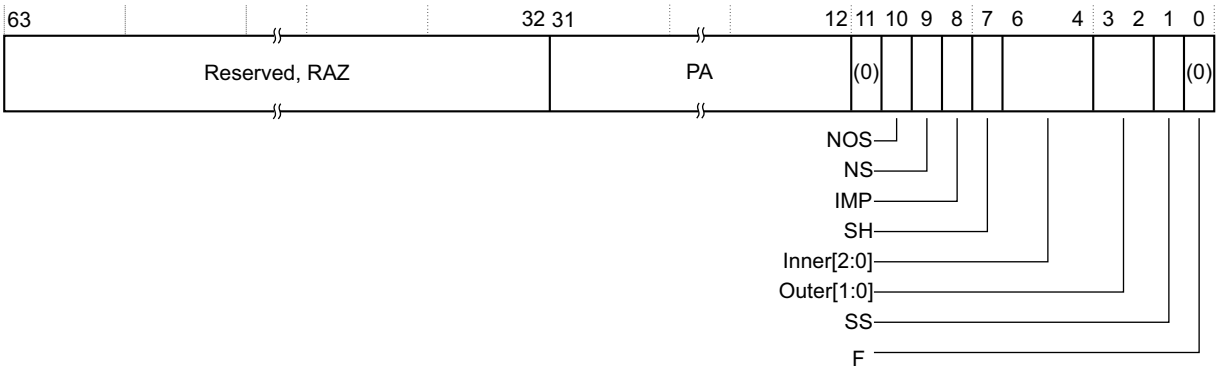
In an implementation that supports two Security states, this register is Banked.

SMMU_NSGPAR is provided as a Secure alias of the Non-secure SMMU_GPAR. See Table 9-1 on page 9-170 and *Secure alias for Non-secure registers on page 9-177*.

Attributes A 64-bit RW register with an UNKNOWN reset value. See also Table 9-1 on page 9-170.

AArch32 Short-descriptor translation scheme

When using the AArch32 Short-descriptor translation scheme, if the translation completes successfully, the format of the SMMU_*s*GPAR bit assignments is:



Bits[63:32] Reserved.

PA, bits[31:12] Bits[31:12] of the physical address.

NOS, bit[10] Not Outer Shareable attribute. Indicates whether the physical memory is Outer Shareable. The possible values of this bit are:

0 Memory is Outer Shareable.

1 Memory is not Outer Shareable.

Whether an implementation distinguishes between Inner Shareable and Outer Shareable memory is IMPLEMENTATION DEFINED. If an implementation does not make this distinction, NOS is UNK/SBZP.

NS, bit[9] Non-secure. The NS attribute for a translation table entry read from a Secure translation context bank.

This bit is UNKNOWN for a translation table entry read from a Non-secure translation context bank.

In SMMU_GPAR, this field is reserved.

IMP, bit[8] IMPLEMENTATION DEFINED.

SH, bit[7] Shareable attribute. Indicates whether the physical memory is shareable. The possible values of this bit are:

0 Physical memory is not shareable.

1 Physical memory is shareable.

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

Inner memory attributes from the translation table entry.

The encoding of this field is:

`0b111` Write-Back, no Write-Allocate.

`0b110` Write-Through.

`0b101` Write-Back, Write-Allocate.

`0b011` Device.

`0b001` Strongly-ordered.

`0b000` Non-cacheable.

All other values are reserved.

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

Outer memory attributes from the translation table. The encoding of this field is:

`0b00` Write-Back, no Write-Allocate.

`0b10` Write-Through, no Write-Allocate.

`0b01` Write-Back, Write-Allocate.

`0b11` Non-cacheable.

SS, bit[1] SuperSection, indicates whether the result is a supersection. The possible values of this bit are:

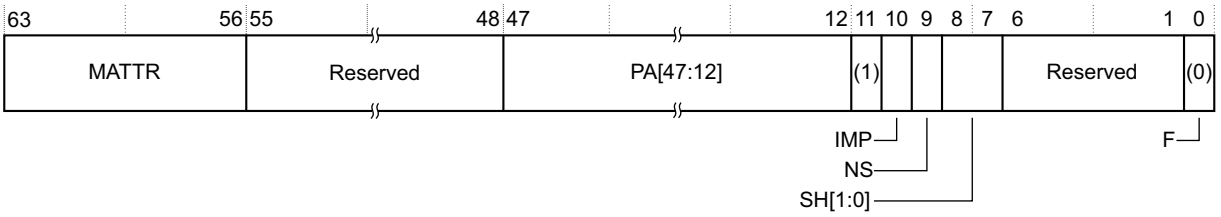
0 The page is not a supersection.
PAR[31:12] contains PAR[31:12] regardless of page size.

1 The page is part of a supersection.
PAR[31:24] contains PA[31:24].
PAR[23:16] contains PA[39:32].
PAR[15:12] contains `0b0000`.

F(0), bit[0] Fault, contains the value 0 on successful completion.

AArch32 Long-descriptor and AArch64 translation schemes

When using either the AArch32 Long-descriptor or the AArch64 translation scheme, if a translation completes successfully, the format of the SMMU_*s*GPAR bit assignments is:



a. In SMMUv1, bits[47:40] are reserved.

MATTR, bits[63:56]

Memory Attributes. These attributes have the encoding of the MAIR field. See *SMMU_CBn_MAIRm, Memory Attribute Indirection Registers on page 16-304* for more information.

Bits[55:48] Reserved.

PA[47:12], bits[47:12]

Bits[47:12] of the physical address.

The number of storage bits is determined by the maximum of SMMU_IDR2.IAS and SMMU_IDR2.OAS. Unimplemented bits are RAZ/WI.

Note

- This field always contains an IPA and is unsigned.
- In SMMUv1, bits[47:40] are reserved.

IMP, bit[10] IMPLEMENTATION DEFINED.

NS, bit[9] Non-secure, the NS attribute for a translation table entry read from a Secure translation context bank.

This bit is UNKNOWN for a translation table entry read from a Non-secure translation context bank.

In SMMU_GPAR, this field is reserved.

SH[1:0], bits[8:7]

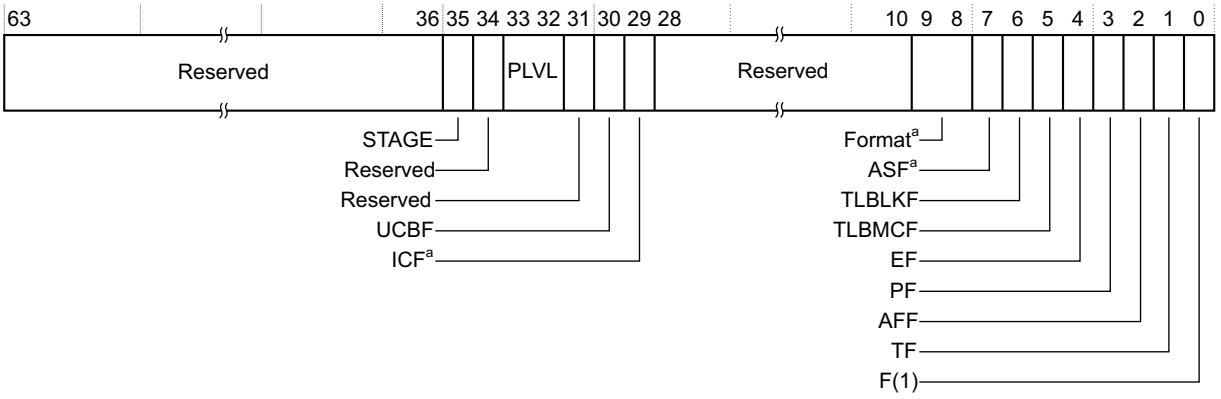
Shareability.

Bits[6:1] Reserved.

F(0), bit[0] Fault, contains the value 0 on successful completion.

Fault format

If a translation fails to complete successfully, the format of the SMMU_*s*GPAR bit assignments, irrespective of the translation format, is:



a. Reserved in SMMUv1

Bits[63:36] Reserved.

STAGE, bit[35]

The stage of translation that encountered the fault. The possible values of this bit are:

0 Fault encountered in stage 1 translation.

1 Fault encountered in stage 2 translation.

Bit[34] Reserved.

PLVL, bit[33:32]

Page Level.

Level of translation table walk that encountered the fault. The possible values of this field are:

`0b00` Level 0. SMMUv2 only.

`0b01` Level 1.

`0b10` Level 2.

`0b11` Level 3.

Bit[31] Reserved.

UCBF, bit[30] Unimplemented context bank fault. The possible values of this bit are:

0 No fault.

1 Fault encountered because an unimplemented context bank was specified.

ICF, Bit[29] Invalid context fault. The possible values of this bit are:

0 No fault.

1 Fault encountered because an invalid context was specified during address translation.

Bits[28:10] Reserved.

Format, bits[9:8]

Translation scheme. This field indicates the translation scheme, and therefore the translation table format for which the recorded fault occurred. The possible values for this field are:

`0b00` AArch32 Short-descriptor translation scheme. This value is not valid for stage 2 translation contexts.

`0b01` AArch32 Long-descriptor translation scheme.

`0b10` AArch64 translation scheme.

`0b11` Reserved. Treated as AArch64 translation scheme.

In SMMUv1, this field is reserved.

Note

This field requires storage.

ASF, bit[7] Address size fault. The possible values for this field are:

0 No fault.

1 Fault is a result of an incorrect address size.

In SMMUv1, this bit is reserved.

TLBLKF, bit[6]

TLB lock fault. The possible values of this bit are:

0 No fault.

1 TLB Lock fault.

Note

In implementations that do not support TLB locking, this bit might be RAZ/WI.

TLBMCF, bit[5]

TLB match conflict fault. The possible values of this bit are:

0 No fault.

1 Fault is a result of multiple matches detected in the TLB.

Note

In implementations that do not support TLB match conflict detection, this bit might be RAZ/WI.

EF, bit[4] External fault. The possible values of this bit are:

0 No fault.

1 An external fault.

Note

In implementations that do not support external fault detection, this bit might be RAZ/WI.

PF, bit[3] Permission fault. The possible values of this bit are:

0 No fault.

1 Fault caused by insufficient permission to complete access.

AFF, bit[2] Access flag fault. The possible values of this bit are:

0 No fault.

1 Access flag fault occurred.

TF, bit[1] Translation fault. The possible values of this bit are:

0 No fault.

1 Invalid mapping for the address being accessed.

F(1), bit[0] Fault.

This bit is set to 1 if the translation aborts.

9.6.21 SMMU_*s*TLBGSTATUS, Global TLB Status register

The SMMU_*s*TLBGSTATUS characteristics are:

Purpose Gives the status of a TLB maintenance operation. See *TLB maintenance operations on page 5-137* for more information.

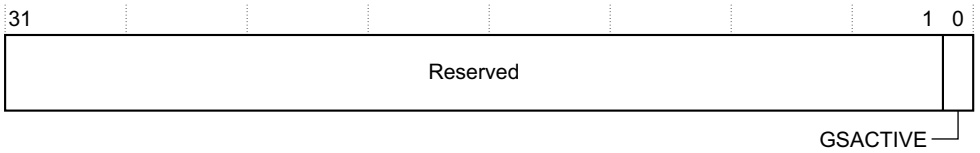
Usage constraints The Security state of the transaction determines whether SMMU_TLBGSTATUS or SMMU_STLBGSTATUS is used.

Configurations In an implementation that supports two Security states, this register is Banked.

SMMU_NSTLBGSTATUS is provided as a Secure alias of the Non-secure SMMU_TLBGSTATUS. See Table 9-1 on page 9-170. See also *Secure alias for Non-secure registers on page 9-177* for general information.

Attributes A 32-bit RO register. See also Table 9-1 on page 9-170.

The SMMU_*s*TLBGSTATUS bit assignments are:



Bits[31:1] Reserved.

GSACTIVE, bit[0]

Global Synchronize TLB Invalidate Active. The possible values of this bit are:

0 No Global TLB synchronization operation is active.

1 A Global TLB synchronization operation is active.

This bit resets to 0.

9.6.22 SMMU_*s*TLBGSYNC, Global Synchronize TLB Invalidate

The SMMU_*s*TLBGSYNC characteristics are:

Purpose Starts a global synchronization operation that ensures the completion of any previously accepted TLB Invalidate operation. As a minimum, the operation applies to the specified Security state, and includes all TLB Invalidate operations initiated in context banks associated with that Security state. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints The Security state of the transaction determines whether SMMU_TLBGSYNC or SMMU_STLBGSYNC is used.

Configurations In an implementation that supports two Security states:

- This register is Banked.
- SMMU_TLBGSYNC only has to apply to Non-secure TLB Invalidate operations.
- SMMU_STLBGSYNC only has to apply to Secure TLB Invalidate operations.

SMMU_NSTLBGSYNC is provided as a Secure alias of the Non-secure SMMU_TLBGSYNC. See Table 9-1 on page 9-170. See also *Secure alias for Non-secure registers on page 9-177* for general information.

Attributes A 32-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_*s*TLBGSYNC bit assignments are reserved.

9.6.23 SMMU_S2CR*n*, Stream-to-Context Register

The SMMU_S2CR*n* characteristics are:

Purpose Specifies an initial context for processing a transaction, where the transaction matches the Stream mapping group that this register belongs to.

Usage constraints An SMMU_S2CR*n* register reserved by Secure software using SMMU_SCR1.NSNUMSMRGO must only specify a translation context bank that is reserved by Secure software.

An SMMU_S2CR*n* register that is accessible from the Non-secure state must only specify a translation context bank that is not reserved by Secure software.

If SMMU_S2CR*n*.MTCFG==1, restrictions apply when the memory is either:

- Strongly-ordered or Device, that is, when MemAttr[3:2]==`0b00`.
- Normal Outer Non-cacheable or Inner Non-cacheable, that is, when MemAttr[3:0]==`0b0101`.

In either of these cases, ARM recommends that:

- RACFG is set to `0b11`, indicating No Read-Allocate.
- WACFG is set to `0b11`, indicating No Write-Allocate.
- TRANSIENTCFG is set to `0b10`, indicating Non-transient.
- SHCFG is set to `0b01`, indicating Outer Shareable.

Configurations The format of this register depends on the state of its TYPE[17:16] field. See:

- *Translation context, type 0b00*.
- *Bypass mode, type 0b01 on page 9-223*.
- *Fault context, type 0b10 on page 9-225*.

See also *Multi-format registers and reserved fields on page 9-180*.

When the Extended Stream Matching Extension is implemented and enabled, the registers map onto the final 128 registers in the Extended Stream Match Register Groups, see Chapter 14 *Extended Stream Matching Extension* for more information.

The number of SMMU_S2CR*n* registers is IMPLEMENTATION DEFINED.

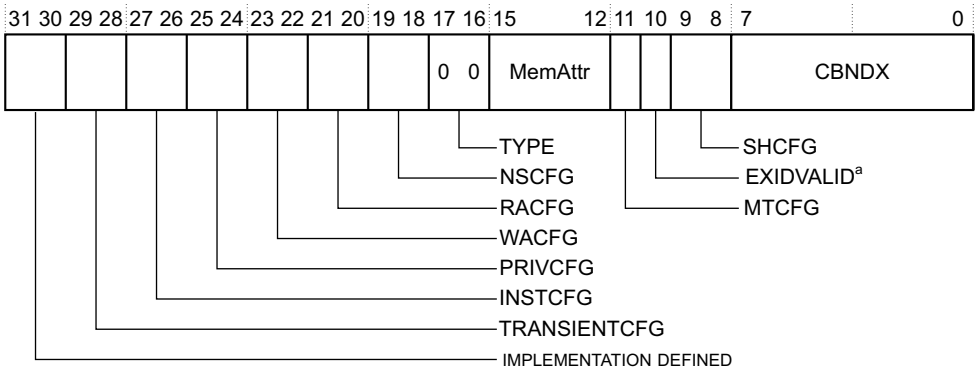
Unimplemented registers are RAZ/WI.

Attributes 32-bit RW registers with UNKNOWN reset values. See also Table 9-1 on page 9-170.

Translation context, type `0b00`

This register format specifies that the initial context is a translation context.

The format of the bit assignments is:



a. Reserved in SMMUv1

Bits[31:30] IMPLEMENTATION DEFINED.

TRANSIENTCFG, bits[29:28]

Transient Allocate Configuration, controls the transient allocation hint.

The encoding of this field is:

0b00 Use the default transient allocation attributes.

0b01 Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

0b10 Non-transient.

0b11 Transient.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, these bits are RAZ/WI.

INSTCFG, bits[27:26]

Instruction Fetch Attribute Configuration. The encoding of this field is:

0b00 Default instruction fetch attribute.

0b01 Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

0b10 Data.

0b11 Instruction.

In SMMUv1, it is IMPLEMENTATION DEFINED whether:

- This field transforms writes.
- The following transactions are restricted by Execute Never permission checking:
  - Incoming write transactions already classified as Instruction.
  - Write transactions that INSTCFG encodes as Instruction

In SMMUv2, this field does not apply to writes. All downstream writes are bus-marked as data.

PRIVCFG, bits[25:24]

Privileged Attribute Configuration. The encoding of this field is:

0b00 Default privilege attributes.

0b01 Meaning depends on the value of SMMU_IDR2.DIPANS:

0 Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

1 Data and Instruction Privileged Access Never.

0b10 Unprivileged.

0b11 Privileged.

Note

In SMMUv2, HYPC and MONC banks are treated as privileged on the downstream bus system, provided the downstream bus system supports such marking. It is IMPLEMENTATION DEFINED whether this attribute is recorded in SMMU_CBn_FSYNR0.PNU.

WACFG, bits[23:22]

Write Allocation Configuration, controls the allocation hint for write accesses. The encoding of this field is:

0b00 Default allocation attributes.

0b01 Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

0b10 Write-Allocate.

0b11 No Write-Allocate.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, these bits are RAZ/WI.

RACFG, bits[21:20]

Read Allocate Configuration, controls the allocation hint for read accesses. The encoding of this field is:

0b00 Default allocation attributes.

0b01 Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Read-Allocate.

`0b11` No Read-Allocate.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, these bits are RAZ/WI.

NSCFG, bits[19:18]

`0b00` Default security attribute.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Secure configuration that only affects Secure SMMU_S2CR*n* entry.

`0b11` Non-secure.

This field exists only for Secure Stream mapping register groups. For Non-secure Stream mapping register groups, it is reserved.

TYPE, bits[17:16]

Initial context type. Indicates the initial context for processing the transaction. The value of this field affects the meaning of the remaining fields in this register. The encoding of this field is:

`0b00` Translation context bank index.

`0b01` Bypass.mode

`0b10` Fault, no index.

`0b11` Reserved. Treated as a fault context.

MemAttr, bits[15:12]

Memory Attributes. See *Memory attribute, MemAttr on page 9-179*.

This field is used only if MTCFG==1.

MTCFG, bit[11]

Memory Type Configuration. The possible values of this bit are:

0 Default memory attributes.

1 MemAttr field attributes.

EXIDVALID, bit[10]

Extended ID Valid. This bit is ignored if SMMU_sCR0.EXIDENABLE==0, otherwise the possible values of this bit are:

0 The Stream Match Register group is invalid.

1 The Stream Match Register group is valid and SMMU_SMRn follows the Extended ID format.

Extended StreamIDs are supported when SMMU_IDR0.EXIDS==1.

In SMMUv1, or when an SMMUV2 implementation does not support 16-bit StreamIDs, that is, when SMMU_IDR0.EXIDS==0, this bit is reserved.

SHCFG, bits[9:8]

Shared Configuration. The encoding of this field is:

`0b00` Default Shareable attribute.

`0b01` Outer Shareable.

`0b10` Inner Shareable.

`0b11` Non-shareable.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, these bits are RAZ/WI.

CBNDX bits[7:0]

Context Bank Index. The translation context bank index for a stage 1 or stage 2 translation. The number of CBNDX bits implemented is IMPLEMENTATION DEFINED. This field must be capable of selecting all of the implemented translation context banks.

An implementation provides the same number of CBNDX bits for every implemented SMMU_S2CRn.

Unimplemented bits behave as RAZ/WI.

In an implementation that supports two Security states, a Secure SMMU_S2CRn register configured to specify a translation context bank is only permitted to:

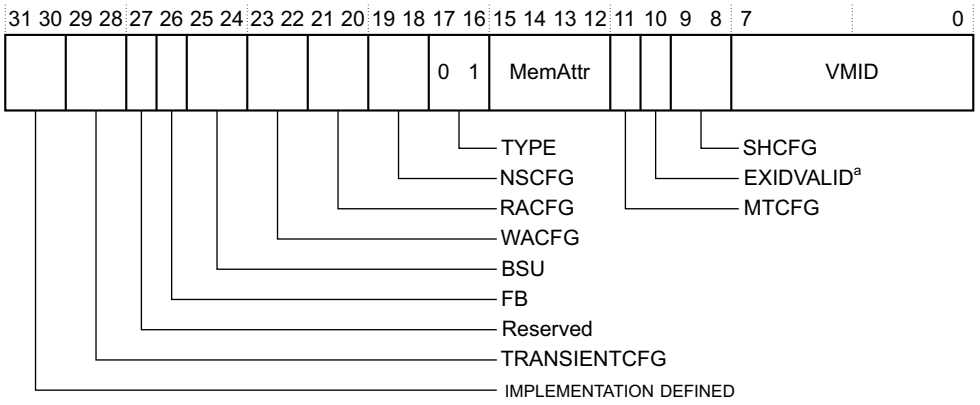
- Specify a CBNDX corresponding to a translation context bank that is also reserved by Secure software.
- Specify a translation context bank configured for the *Stage 1 context with stage 2 bypass, TYPE==0b01 on page 10-242* format.
- Similarly, in an implementation that supports two Security states, a Non-secure SMMU_S2CRn register is only permitted to specify a translation context bank that is reserved for use by the Non-secure software.

Bypass mode, type `0b01`

This register format specifies that Bypass mode is to be used.

No address translation is applied. Memory attributes can be specified to overlay or override those associated with the transaction.

The format of the bit assignments is:



a. Reserved in SMMUv1

Bits[31:30] IMPLEMENTATION DEFINED.

TRANSIENTCFG, bits[29:28]

Transient Allocate Configuration.

The encoding of this field is:

`0b00` Use default transient allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Non-transient.

`0b11` Transient.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, it is RAZ/WI.

Bit[27] Reserved.

FB, bits[26] Force Broadcast. Force Broadcast of TLB, branch predictor, and instruction cache maintenance operations.

This field affects client TLB maintenance, BPIALL and ICIALLU operations. If it has the value 1, any affected operation is modified to the equivalent broadcast variant within the Inner Shareable domain.

The possible values of this bit are:

0 Process affected operations as presented.

1 Upgrade affected operations to be broadcast within the Inner Shareable domain.

BSU, bits[25:24]

Barrier Shareability Upgrade.

This field upgrades the required shareability domain of barriers issued by client devices mapped to this Stream mapping register group by setting the minimum shareability domain applied to any barrier.

The encoding of this field is:

`0b00` No effect.

`0b01` Inner Shareable.

`0b10` Outer Shareable.

`0b11` Full system.

Upgrade of the barrier shareability domain might not be supported in all system topologies. In an implementation that does not have this upgrade behavior, this field is RAZ/SBZP.

WACFG, bits[23:22]

Write-Allocate Configuration, an allocation hint for write accesses. The encoding of this field is:

`0b00` Default allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Write-Allocate.

`0b11` No Write-Allocate.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, these bits are RAZ/WI.

RACFG, bits[21:20]

Read-Allocate Configuration. Gives an allocation hint for read accesses. The encoding of this field is:

`0b00` Default allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Read-Allocate.

`0b11` No Read-Allocate.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, these bits are RAZ/WI.

NSCFG, bits[19:18]

`0b00` Default security attribute.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Secure configuration that only affects Secure SMMU_S2CR*n* entry.

`0b11` Non-secure.

This field exists only for Secure Stream mapping register groups. For Non-secure Stream mapping register groups, it is reserved.

TYPE, bits[17:16]

Register type. Indicates the meaning of the remaining fields in this register. The encoding of this field is:

`0b00` Translation context bank index.

`0b01` Bypass mode.

`0b10` Fault, no index.

`0b11` Reserved. Treated as a fault context.

MemAttr, bits[15:12]

Memory Attributes. See *Memory attribute, MemAttr on page 9-179*.

This field is used only if MTCFG==1.

MTCFG, bit[11]

Memory Type Configuration. The possible values of this bit are:

0 Default memory attributes.

1 MemAttr field attributes.

EXIDVALID, bit[10]

Extended ID Valid. This bit is ignored if SMMU_sCR0.EXIDENABLE==0, otherwise the possible values of this bit are:

0 The Stream Match Register group is invalid.

1 The Stream Match Register group is valid and SMMU_SMRn follows the Extended ID format.

Extended StreamIDs are supported when SMMU_IDR0.EXIDS==1.

In SMMUv1, or when an SMMUV2 implementation does not support 16-bit StreamIDs, that is, when SMMU_IDR0.EXIDS==0, this bit is reserved.

SHCFG, bits[9:8]

Shared Configuration. The encoding of this field is:

`0b00` Default Shareable attribute.

`0b01` Outer Shareable.

`0b10` Inner Shareable.

`0b11` Non-shareable.

It is IMPLEMENTATION DEFINED whether this field is present. If not present, these bits are RAZ/WI.

VMID, bits[7:0]

Reserved for IMPLEMENTATION DEFINED use of a VMID field, where such use is relevant to all transactions, including those not subject to any translation.

It is IMPLEMENTATION DEFINED whether this field is implemented, and what effect it has.

In an implementation that does not include this behavior, this field is RAZ/WI.

If an implementation that supports two Security states does not support a VMID for Secure translation regimes, any Secure VMID field is ignored. See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

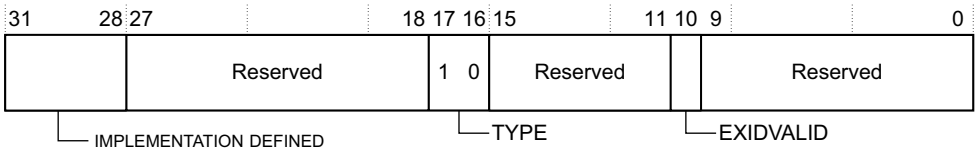
Note

This field remains as 8 bits even if 16-bit VMIDs are supported when using MMU translations, that is, SMMU_IDR2.VMID16S is one.

Fault context, type `0b10`

This format specifies that the Fault context is to be used. Any transaction that maps to this Stream mapping group incurs an invalid context fault.

The format of the bit assignments is:



Bits[31:28] IMPLEMENTATION DEFINED.

Bits[27:18] Reserved.

TYPE, bits[17:16]

Register Type. Indicates the meaning of the remaining fields in this register. The encoding of this field is:

`0b00` Translation context bank index.

`0b01` Bypass mode.

`0b10` Fault, no index.

`0b11` Reserved.

Bits[15:11] Reserved.

EXIDVALID, bit[10]

Extended ID Valid. This bit is ignored if SMMU_sCR0.EXIDENABLE==0, otherwise the possible values of this bit are:

0 The Stream Match Register group is invalid.

1 The Stream Match Register group is valid and SMMU_SMRn follows the Extended ID format.

Extended StreamIDs are supported when SMMU_IDR0.EXIDS==1.

In SMMUv1, or when an SMMUV2 implementation does not support 16-bit StreamIDs, that is, when SMMU_IDR0.EXIDS==0, this bit is reserved.

Bits[9:0] Reserved.

9.6.24 SMMU_SCR1, Secure Configuration Register 1

The SMMU_SCR1 characteristics are:

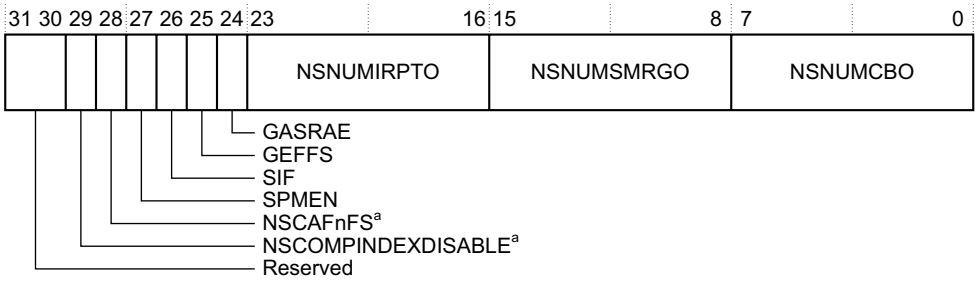
Purpose Provides top-level Secure control of the SMMU.

Usage constraints Secure access only.

Configurations Only present in an implementation that supports two Security states.

Attributes A 32-bit RW register. See the field descriptions for information about the reset values. See also Table 9-1 on page 9-170.

The SMMU_SCR1 bit assignments are:



a. Reserved in SMMUv1

Bits[31:30] Reserved.

NSCOMPINDEXDISABLE, bit[29]

Non-secure Compressed Index Disable. The possible values of this bit are:

0 The SMMU_COMPINDEXn registers are assigned for Non-secure use, the StreamID Compressed Indexing Extension is available only to Non-secure transactions.

1 The SMMU_COMPINDEXn registers are assigned for Secure use, the StreamID Compressed Indexing Extension is available only to Secure transactions.

This bit resets to 0.

In SMMUv1, this bit is RES0.

In SMMUv2, this bit is RES0 if the StreamID Compressed Indexing Extension is not implemented. See Chapter 15 *StreamID Compressed Indexing Extension* for more information.

NSCAFnFS, bit[28]

Non-secure configuration access faults not forced Secure. The possible values of this bit are:

0 Configuration access faults caused by Non-secure accesses to Secure registers are reported using:

- SMMU_SGFSR as the Fault Status Register.
- SMMU_SGFSYNR0 and SMMU_SGFSYNR1 as the Syndrome Registers.
- SMMU_SGFAR as the Fault Address Register.
- SMMU_gCfgIrpt as the reporting interrupt.

1 Configuration access faults caused by Non-secure accesses to Secure registers are reported using:

- SMMU_NSGFSR as the Fault Status Register.
- SMMU_NSGFSYNR0 and SMMU_NSGFSYNR1 as the Syndrome Registers.
- SMMU_NSGFAR as the Fault Address Register.
- SMMU_NSgCfgIrpt as the reporting interrupt.

If SMMU_SCR1.NSCAFRO==1, all configuration access faults caused by Non-secure accesses that would have been recorded in SMMU_SGFSR are instead recorded in SMMU_GFSR. See *SMMU_sGFSR, Global Fault Status Register on page 9-209*.

Generation of configuration access faults is an IMPLEMENTATION DEFINED feature. If an implementation can generate configuration access faults, the implementation might either make this bit programmable or a fixed, IMPLEMENTATION DEFINED value. If this bit is programmable, it resets to zero. If an implementation cannot generate configuration access faults, this bit is RAZ/WI.

This bit does not apply to configuration access faults caused by Non-secure accesses to Common or Non-secure registers.

In SMMUv1, this bit is reserved.

Note

In implementations where this bit has an effect and is set to 0, Secure software permits Non-secure software to report faults to the Secure state. This might interfere with the use of SMMU_SGFSR by Secure software.

SPMEN, bit[27]

Secure Performance Monitor Enable. The possible values of this bit are:

0 Any event caused by Secure transaction processing does not contribute towards performance monitor counting.

1 Any event caused by Secure transaction processing is permitted to contribute towards performance monitor counting.

This bit resets to 0.

SIF, bit[26] Secure Instruction Fetch. The possible values of this bit are:

0 Secure instruction fetches are permitted to Non-secure memory locations.

1 Raise a permission fault if a Secure domain access attempts to exit the SMMU as a Non-secure instruction.

See *Secure Instruction Fetch (SIF) permission faults on page 3-102* for more information.

This bit resets to 0.

Note

If a transaction is associated with a particular translation context bank, faults are recorded in SMMU_CBn_FSR. Otherwise they are recorded in SMMU_SGFSR.

GEFFS, bit[25]

Global external faults forced Secure. The possible values of this bit are:

0 Permit SMMU_GFSR to report external faults.

1 SMMU_SGFSR reports all external faults.

If SMMU_SCR1.GEFRO==1, all external aborts that would have been recorded in SMMU_GFSR are instead recorded in SMMU_SGFSR. See *SMMU_sGFSR, Global Fault Status Register on page 9-209*.

This bit resets to zero.

In SMMUv1, this bit resets to an UNKNOWN value.

GASRAE, bit[24]

Global Address Space Restricted Access Enable. The possible values of this bit are:

0 The Global address space has default access permission, permitting Secure and Non-secure configuration memory accesses.

1 The Global address space is only accessible by Secure configuration memory accesses. Stage 2 format context banks are accessible only by Secure configuration accesses.

Whether SMMU_SCR1.GASRAE==1 affects the IMPLEMENTATION DEFINED registers in Global Register space 0 at offsets `0xFD0 - 0xFFC`, that may contain ID codes, is IMPLEMENTATION DEFINED.

Whether SMMU_SCR1.GASRAE==1 affects the IMPLEMENTATION DEFINED address space is IMPLEMENTATION DEFINED.

UNPREDICTABLE behavior results if Secure software does not follow these constraints.

This bit resets to 0.

NSNUMIRPTO, bits[23:16]

Non-secure Number of Interrupts Override.

In SMMUv1, it is IMPLEMENTATION DEFINED whether this field is fully implemented with writable storage. A portion of this field might behave as RAZ/WI because the implementation might not provide a sufficient number of interrupts for some of the upper bits of NSNUMIRPTO to be relevant.

In implementations that do not support Secure translation contexts, such as those that support stage 2 translation format only, it is also IMPLEMENTATION DEFINED whether this field is read-only. In such implementations, Secure software does not manage translation contexts and therefore there is no requirement to reserve context interrupts for Secure use.

These bits reset to the implemented number of interrupts. See *SMMU_IDR0-7, Identification registers on page 9-182*.

For more information, see:

- *Resource allocation on page 7-158*.
- *SMMU_IDR0 on page 9-182*.

In SMMUv2, the interpretation is complex. See *Context interrupts on page 3-100* for more information.

NSNUMSMRGO, bits[15:8]

Non-secure Number of Stream Mapping Register Groups Override.

Adjusts the number of Stream mapping register groups visible to Non-secure accesses. The number of Stream mapping register groups reported in SMMU_IDR0 is reduced to the number indicated by SMMU_SCR1.NSNUMSMRGO. See *SMMU_IDR0-7, Identification registers on page 9-182*.

These bits reset to the implemented number of Stream mapping register groups. See *SMMU_IDR0-7, Identification registers on page 9-182*.

Note

If the value in NSNUMSMRGO exceeds the number of implemented Stream Match Register groups, Non-secure software might attempt to access an unimplemented Stream Match Register group. It is IMPLEMENTATION DEFINED whether Non-secure accesses to unimplemented Stream Match Register groups result in configuration access faults or are ignored.

It is IMPLEMENTATION DEFINED whether this field is fully implemented with writable storage. A portion of this field might be RAZ/WI because the implementation might not provide sufficient Stream mapping register groups for some of the upper bits of SMMU_SCR1.NSNUMSMRGO to be relevant.

These bits reset to the implemented number of stream mapping register groups.

In an implementation that includes the Extended Stream Matching Extension, this field resets to 128.

For more information, see:

- *The Stream mapping table on page 2-54*.
- *Resource allocation on page 7-158*.
- *Permitted transaction resource usage on page 7-159*.
- *SMMU_IDR0 on page 9-182*.

NSNUMCBO, bits[7:0]

Non-secure Number of Context Banks Override.

Specifies the number of translation context banks visible to Non-secure accesses. For Non-secure accesses, the number of translation context banks reported in SMMU_IDR1.NUMCB is reduced to the number indicated by SMMU_SCR1.NSNUMCBO.

Note

Allocating translation context banks for Non-secure use includes the SMMU_CBARn registers and SMMU_CBFRSYNRAn registers associated with the affected translation context banks.

This field resets to the physically implemented number of translation context banks.

In SMMUv2, a change to this field sets the corresponding SMMU_CBA2Rn.VA64 bit of the reallocated context banks to an UNKNOWN state.

Note

A consequence of this is that SMMUv1 Secure software must be updated to always set SMMU_CBA2Rn.VA64 to 0 when it changes SMMU_SCR1.NSNUMCBO. See *Resource allocation on page 7-158* for more information.

This field must not be set to a value below that reported by SMMU_IDR1.NUMS2CB. Stage 2 format translation is not available to Secure transactions.

It is IMPLEMENTATION DEFINED whether this field is fully implemented with writable storage. A portion of this field might behave as RAZ/WI. This is permitted because the implementation might not provide sufficient contexts for some of the upper bits of SMMU_SCR1.NSNUMCBO to be relevant.

In implementations that do not support Secure translation contexts, such as those that support stage 2 translation format only, it is also IMPLEMENTATION DEFINED whether this field is read-only. In such implementations, Secure software does not manage translation contexts and there is therefore no requirement to reserve context interrupts for Secure use.

In SMMUv2, if a context bank has a stalled transaction when the corresponding SMMU_CBARn, or SMMU_CBA2Rn is written, an implementation might retry the stalled transaction immediately after SMMU_CBARn or SMMU_CBA2Rn is updated. An implementation might also retry a transaction in all context banks affected by a change to SMMU_SCR1.NSNUMCBO. See *The Stall fault model on page 3-107* for more information.

For more information, see:

- *The translation context bank table on page 2-75*.
- *The Context Bank Attribute Register, SMMU_CBARn on page 2-78*.
- *Recording a context fault on page 3-102*.
- *Resource allocation on page 7-158*.
- *SMMU_IDR1 on page 9-184*.

9.6.25 SMMU_SMR*n*, Stream Match Register

The SMMU_SMR*n* characteristics are:

Purpose Matches a transaction with a particular Stream mapping register group.

Usage constraints During configuration, the Stream Match Register table can have multiple entries that match the same Stream Identifier value, possibly resulting in UNPREDICTABLE behavior. See *StreamID matching on page 2-57* for more information about multiple matches. To prevent multiple matches, software must ensure that no transactions that might match the StreamID are received, by:

- Disabling any source client devices that might match.
- Ensuring that no outstanding transactions from these client devices are in progress.

As an extra precaution, software can first disable all affected SMMU_SMR*n* table entries by setting the SMMU_SMR*n*.VALID bit to 0, then reprogramming the entries as appropriate.

For every implemented Stream Match Register, an implementation must provide:

- The same number of ID bits and MASK bits.
- The same number of EXID bits and EXMASK bits.

It is IMPLEMENTATION DEFINED whether ID or EXID bits are preserved when the corresponding MASK or EXMASK bit is set to 1 to indicate that the corresponding bit of ID or EXID is ignored in matching).

Configurations This register cannot be implemented in SMMUv1.

The number of SMMU_SMR*n* registers is IMPLEMENTATION DEFINED.

In an implementation that includes StreamID matching, each of these registers has RW access. In an implementation that includes StreamID indexing, each register is RAZ/WI.

The format of this register depends on the value of the SMMU_sCR0.EXIDENABLE bit.

When the Extended Stream Matching Extension is implemented and enabled, the registers map onto the final 128 registers in the Extended Stream Match Register Groups, see Chapter 14 *Extended Stream Matching Extension* for more information.

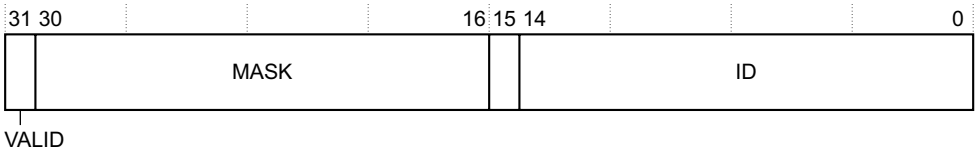
Attributes 32-bit RW registers with access attributes that depend on the configuration of the implementation. See the configuration details for information.

This register resets to an UNKNOWN value.

See also Table 9-1 on page 9-170.

SMMU_SMRn bit assignments for SMMUv1, or when SMMU_sCR0.EXIDENABLE==0

The SMMU_SMR*n* bit assignments are:



VALID, bit[31]

The possible values of this bit are:

0 Entry is not included in the Stream mapping table search.

1 Entry is included in the Stream mapping table search.

MASK, bits[30:16]

Masking of StreamID bits irrelevant to the matching process:

- If MASK[i]==1, ID[i] is ignored.
- If MASK[i]==0, ID[i] is relevant for matching.

The actual number of MASK bits is IMPLEMENTATION DEFINED. Unimplemented bits are RAZ/WI.

Unless an SMMU implementation has a particular requirement for non-contiguous MASK bits, ARM recommends that:

- The MASK bits are contiguous.
- The number of MASK bits implemented is [SMMU_IDR0.NUMSIDB - 1].

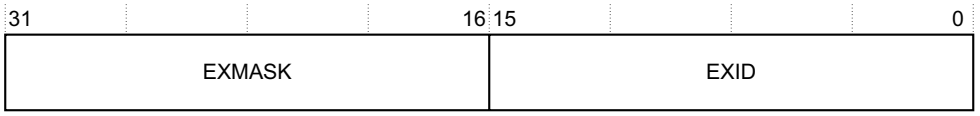
Bit[15] Reserved.

ID, bits[14:0] The Stream Identifier to match.

The actual number of ID bits is IMPLEMENTATION DEFINED. Unimplemented bits are RAZ/WI.

SMMU_SMRn bit assignments for SMMUv2, when SMMU_sCR0.EXIDENABLE==1

The SMMU_SMR*n* bit assignments are:



EXMASK, bits[31:16]

Extended Mask. Masking of StreamID bits irrelevant to the matching process:

- If EXMASK[i]==1, EXID[i] is ignored.
- If EXMASK[i]==0, EXID[i] is relevant for matching.

EXID, bits[15:0]

Extended ID. The Stream Identifier to match.

The entry is valid if the value of the corresponding SMMU_S2CRn.EXIDVALID bit is 1.

Note

When this description relates to an SMMU_EXSMRn, the entry is valid if the value of the corresponding SMMU_EXS2CRn.EXIDVALID bit is 1.

9.6.26 SMMU_STLBIALL, TLB Invalidate All

The SMMU_STLBIALL characteristics are:

Purpose Invalidates all unlocked Secure entries in the TLB. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked Secure TLB entries. That is, it must invalidate all TLB entries that are marked as Secure, including MONC TLB entries. However it is IMPLEMENTATION DEFINED whether it invalidates Secure Hypervisor entries.

Note

Because an SMMU_STLBIALL operation might not invalidate TLB entries allocated in the Secure HYPC bank, a separate write to SMMU_TLBIALLH is required to ensure that these entries are invalidated.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries.

Configurations In an implementation that does not support two Security states, this register is reserved. RAZ/WI.

Attributes A 32-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_STLBIALL bit assignments are reserved.

9.6.27 SMMU_STLBIALLM, Secure TLB Invalidate All MONC

The SMMU_STLBIALLM characteristics are:

Purpose Invalidates all unlocked TLB entries associated with MONC banks. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked MONC entries.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries.

Configurations This register is not provided in SMMUv1.

In an implementation that does not support two Security states, this register is reserved, RAZ/WI.

Attributes A 32-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_STLBIALLM bit assignments are reserved.

9.6.28 SMMU_TLBIALLH, TLB Invalidate All Hyp

The SMMU_TLBIALLH characteristics are:

Purpose Invalidates all Hyp-tagged entries in the TLB. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked Hyp-tagged entries.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries.

Configurations None.

Attributes A 32-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_TLBIALLH bit assignments are reserved.

9.6.29 SMMU_TLBIALLNSNH, TLB Invalidate All Non-secure Non-Hyp

The SMMU_TLBIALLNSNH characteristics are:

Purpose Invalidates all TLB entries that are tagged as Non-secure non-Hyp. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked Non-secure non-Hyp TLB entries.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries.

Configurations None.

Attributes A 32-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_TLBIALLNSNH bit assignments are reserved.

9.6.30 SMMU_STLBIVALM, Secure TLB Invalidate MONC by VA, Last level

The SMMU_STLBIVALM characteristics are:

Purpose Invalidates all unlocked TLB entries associated with MONC banks, that match the specified virtual address and hold information from the final level of lookup. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints 32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

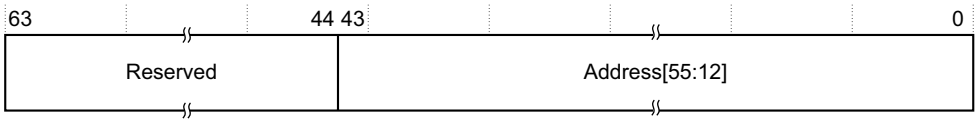
An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries.

Configurations This register is not provided in SMMUv1.

In an implementation that does not support two Security states, this register is reserved, RAZ/WI.

Attributes A 64-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_STLBIVALM bit assignments are:



Bits[63:44] Reserved.

Address[55:12], bits[43:0]

Bits[55:12] of the address to be invalidated.

- If the translation granule is 64KB, then bits corresponding to Address[15:12] are ignored.
- If the translation granule is 16KB, then bits corresponding to Address[13:12] are ignored.
- The address extends to bit[63] by copying bit[55].

9.6.31 SMMU_STLBIVAM, Secure TLB Invalidate MONC by VA

The SMMU_STLBIVAM characteristics are:

Purpose Invalidates all unlocked TLB entries associated with MONC banks, that match the specified virtual address. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints 32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

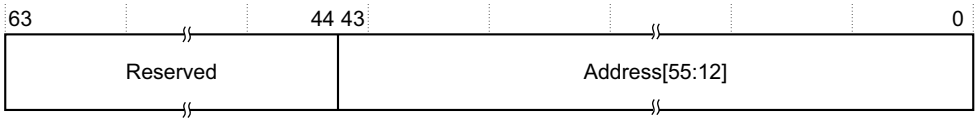
An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations This register is not provided in SMMUv1.

In an implementation that does not support two Security states, this register is reserved, RAZ/WI.

Attributes A 64-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_STLBIVAM bit assignments are:



Bits[63:44] Reserved.

Address[55:12], bits[43:0]

Bits[55:12] of the address to be invalidated.

- If the translation granule is 64KB, then bits corresponding to Address[15:12] are ignored.
- If the translation granule is 16KB, then bits corresponding to Address[13:12] are ignored.

- The address extends to bit[63] by copying bit[55].

9.6.32 SMMU_TLBIVAH, Invalidate Hyp TLB by VA

The SMMU_TLBIVAH characteristics are:

Purpose Invalidates all unlocked Hyp TLB entries that match the specified virtual address. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked Hyp-tagged TLB entries associated with the specified virtual address.

In an SMMU implementation that does not support the AArch32 translation schemes, this register is reserved, RAZ/WI.

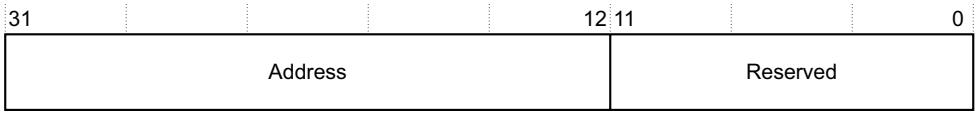
An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations None.

Attributes A 32-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_TLBIVAH bit assignments are:



Address, bits[31:12]

Bits[31:12] of the VA to invalidate.

Bits[11:0] Reserved.

9.6.33 SMMU_TLBIVAH64, Invalidate Hyp TLB by VA, AArch64

The SMMU_TLBIVAH64 characteristics are:

Purpose Invalidates all Hyp TLB entries that match the specified virtual address. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked Hyp-tagged TLB entries associated with the specified virtual address.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

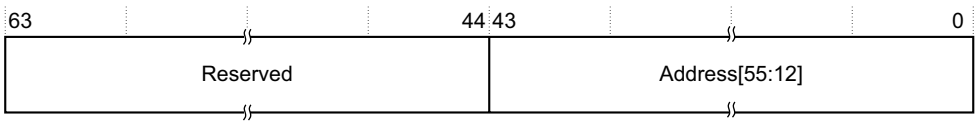
An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations This register is not provided in SMMUv1.

Attributes A 64-bit WO register.

See also Table 9-1 on page 9-170.

The SMMU_TLBIVAH64 bit assignments are:



Bits[63:44] Reserved.

Address[55:12], bits[43:0]

Bits[55:12] of the address to be invalidated.

- If the translation granule is 64KB, then bits corresponding to Address[15:12] are ignored.
- If the translation granule is 16KB, then bits corresponding to Address[13:12] are ignored.
- The address extends to bit[63] by copying bit[55].
- For AArch32 translation schemes, bits[55:32] are treated as zero.

9.6.34 SMMU_TLBIVALH64, Invalidate Hyp TLB by VA, Last level, AArch64

The SMMU_TLBIVALH64 characteristics are:

Purpose Invalidates Hyp TLB entries that match the specified virtual address and hold information from the final level of lookup. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked Hyp-tagged TLB entries that match the specified virtual address and hold information from the final level of lookup.

32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations This register is not provided in SMMUv1

Attributes A 64-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_TLBIVALH64 bit assignments are the same as those for SMMU_TLBIVAH64.

9.6.35 SMMU_TLBIVMID, TLB Invalidate by VMID

The SMMU_TLBIVMID characteristics are:

Purpose Invalidates all Non-secure non-Hyp TLB entries that have the specified VMID. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked Non-secure non-Hyp TLB entries that have the specified VMID.

This register does not affect Secure TLB entries.

Note

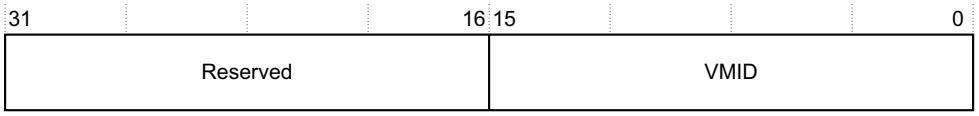
In implementations that support a VMID for Secure context banks, TLB entries created by Secure banks are not tagged with the VMID. This operation does not affect such entries. See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations None.

Attributes A 32-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_*s*TLBIVMID bit assignments are:



Bits[31:16] Reserved.

VMID, bits[15:0] The Virtual Machine Identifier to use in the Invalidate operation.

When SMMU_IDR2.VMID16S is zero, then bits[15:8] are Reserved.

Bits[15:8] are Reserved in SMMUv1.

See *VMID size on page 1-39*.

9.6.36 SMMU_TLBIVMIDS1, TLB Invalidate Stage 1 by VMID

The SMMU_TLBIVMIDS1 characteristics are:

Purpose Invalidates all Non-secure non-Hyp entries that hold information from stage 1 translations with the specified VMID. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked stage 1 Non-secure non-Hyp TLB entries that have the specified VMID.

This register does not affect Secure TLB entries.

In an implementation that supports stage 1 followed by stage 2 translation, this register must invalidate any entries tagged with specified VMID.

Note

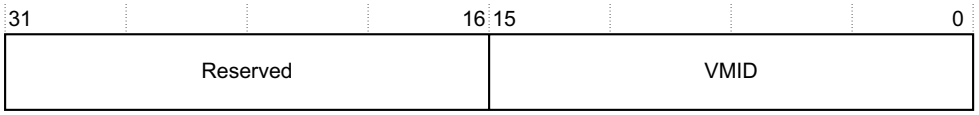
In implementations that support a VMID for Secure context banks, TLB entries created by Secure banks are not tagged with the VMID. This register does not affect such entries. See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations This register is not provided in SMMUv1.

Attributes A 32-bit WO register. See also Table 9-1 on page 9-170.

The SMMU_*s*TLBIVMIDS1 bit assignments are:



Bits[31:16] Reserved.

VMID, bits[15:0]

The Virtual Machine Identifier to use in the Invalidate operation.

If SMMU_IDR2.VMID16S is zero, then bits[15:8] are Reserved. See *VMID size on page 1-39*.

Chapter 10
SMMU Global Register Space 1

This chapter defines SMMU Global Register Space 1. It contains the following sections:

10.1 SMMU Global Register Space 1 register summary

SMMU Global Register Space 1 provides high-level SMMU resource control, and accommodates the number of addresses in the global register space exceeding the capacity of a single memory page, when the page size is 4KB. The size of this address space is defined by PAGESIZE. See *PAGESIZE and NUMPAGENDXB on page 8-164*.

The SMMU architecture supports atomic register access at the size of the register. Whether atomic register accesses of other sizes are supported is IMPLEMENTATION DEFINED.

Table 10-1 shows the register space relative to the offset from SMMU_GR1_BASE.

Table 10-1 SMMU Global Register Space 1

Offset	Name	Width	Type	Description
0x00000	SMMU_CBAR0	32-bit	RW	*SMMU_CBARn, Context Bank Attribute Registers on page 10-241*
0x00004	SMMU_CBAR1			
0x00008-0x001FC	SMMU_CBAR2 to SMMU_CBAR127			
0x00200-0x003FC	Reserved	-	-	-
0x00400	SMMU_CBFRSYNRA0	32-bit	RW	*SMMU_CBFRSYNRAn, Context Bank Fault Restricted Syndrome Register A on page 10-249*
0x00404	SMMU_CBFRSYNRA1			
0x00408-0x005FC	SMMU_CBFRSYNRA2 to SMMU_CBFRSYNRA127			
0x00600-0x007FC	Reserved	-	-	-
0x00800	SMMU_CBA2R0	32-bit	RW	*SMMU_CBA2Rn, Context Bank Attribute Registers on page 10-247*
0x00804	SMMU_CBA2R1			
0x00808-0x009FC	SMMU_CBA2R2 to SMMU_CBA2R127			
0x00A00-(PAGESIZE−0x4)	Reserved	-	-	-

10.2 SMMU Global Register Space 1 register descriptions

This section describes all of the Global Register Space 1 registers that might be present in an SMMU implementation. Registers are shown in register name order.

10.2.1 SMMU_CBAR*n*, Context Bank Attribute Registers

The SMMU_CBAR*n* characteristics are:

Purpose When SMMU_S2CRn specifies that the initial context for a transaction is a translation context bank, then the SMMU_CBARn registers specify configuration attributes for translation context bank *n*.

Usage constraints SMMU_IDR1.NUMS2CB specifies the IMPLEMENTATION DEFINED number of translation context banks that only support stage 2 translation. See *The translation context bank table on page 2-75* for more information about discovering whether an implementation uses such context banks.

In an implementation that supports two Security states, Secure context banks must use TYPE `0b01`, see *Stage 1 context with stage 2 bypass, TYPE==0b01 on page 10-242*. Using any other TYPE generates an invalid context fault if a transaction attempts to use this bank.

If a context bank has a stalled transaction, indicated by SMMU_CBn_FSR.SS==1, when the corresponding SMMU_CBARn register is written, an implementation can retry the stalled transaction immediately after updating SMMU_CBAR. That is, on a write to the SMMU_CBARn register for a stalled transaction, an implementation can behave as if the value of 0 is written to SMMU_CBn_RESUME.

If SMMU_CBAR*n*.TYPE selects a stage of translation that is not supported by any of SMMU_IDR0.{NTS, S1TS, S2TS}, then an Invalid context fault is generated if a transaction attempts to use this bank.

If a context bank only supports a single stage of translation, either because the implementation only supports a single stage of translation, or because the context bank is implemented as a stage 2 only context bank, then the SMMU_CBAR*n*.TYPE field can be a read-only field that returns the appropriate TYPE value.

Configurations The format of this register depends on the value of its TYPE field. *SMMU_CBARn format* gives links to the descriptions of the different formats.

The number of SMMU_CBAR*n* registers is IMPLEMENTATION DEFINED.

See also *Multi-format registers and reserved fields on page 9-180*.

Attributes 32-bit RW registers. See also Table 10-1 on page 10-240.

SMMU_CBAR*n* format

The SMMU_CBAR*n*.TYPE field indicates the translation context that this context bank provides for the transaction. The value of this field affects the meaning of the remaining SMMU_CBARn fields. The encoding of SMMU_CBAR*n*.TYPE is:

`0b00` Stage 2 context. The context bank provides only stage 2 translation. See *Stage 2 context, TYPE==0b00 on page 10-242*.

`0b01` Stage 1 context with stage 2 bypass. The context bank provides stage 1 translation, and stage 2 memory attribute-only transformation. See *Stage 1 context with stage 2 bypass, TYPE==0b01 on page 10-242*.

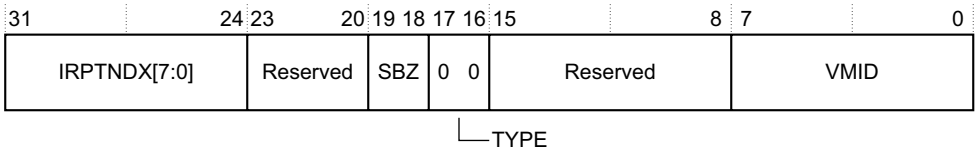
`0b10` Stage 1 context with stage 2 fault. The context bank generates an invalid context fault. See *Stage 1 context with stage 2 fault, TYPE==0b10 on page 10-246*.

`0b11` Stage 1 followed by stage 2 translation context. The context bank provides stage 1 translation followed by stage 2 translation, and specifies an additional translation context bank for the stage 2 translation. See *Stage 1 followed by stage 2 translation context, TYPE==0b11 on page 10-246*.

An SMMU implementation can configure the SMMU_CBAR*n*.TYPE field to be read-only. For example, in an implementation that supports only stage 2 translation, this could hold a read-only value of `0b00`, removing any requirement to implement storage for this field.

Stage 2 context, TYPE==`0b00`

The format of the bit assignments is:



IRPTNDX[7:0], bits[31:24]

Interrupt Index.

In SMMUv1, this provides the Context interrupt number to assert in the event of an interrupt raising a fault in the associated translation context bank. SMMU_IDR0.NUMIRPT specifies the range of values that software can configure this field in.

It is IMPLEMENTATION DEFINED whether this field is fully implemented with writable storage. See *Context interrupts on page 3-100* for more information.

In SMMUv2, each context bank has a dedicated interrupt pin, and there is therefore no requirement for a Context interrupt number. See *Context interrupts on page 3-100* for more information.

Bits[23:20] Reserved.

SBZ, bits[19:18]

Should-Be-Zero.

TYPE, bits[17:16]

Translation context type. See *SMMU_CBARn format on page 10-241* for more information.

Bits[15:8] Reserved.

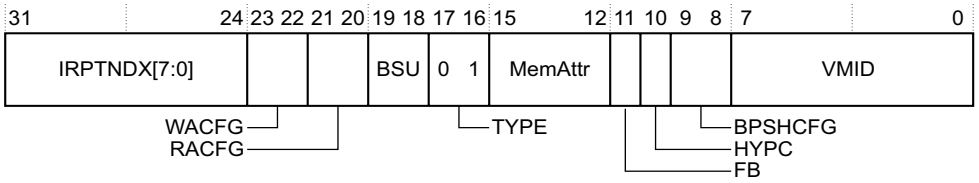
VMID, bits[7:0]

When the value of SMMU_CR0.VMID16EN is 0, indicating the use of 8-bit VMID, this field holds the Virtual Machine Identifier (VMID) to be associated with the translation context bank. If SMMU_CR0.VMID16EN is 1, then this field is RES0, and the VMID comes from SMMU_CBA2Rn.VMID16. See *VMID size on page 1-39*.

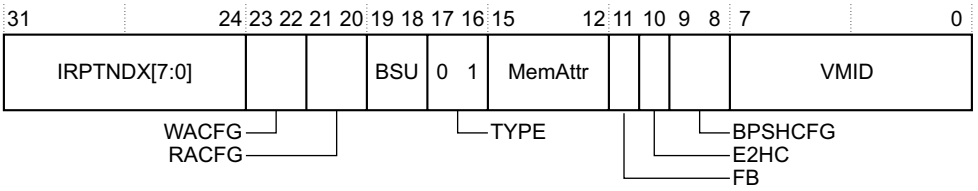
Stage 1 context with stage 2 bypass, TYPE==`0b01`

For TYPE == `0b01`, in an SMMUv2 implementation that supports E2HC, the assignment of bit[10] depends on the value of SMMU_CR0.HYPMODE. If an implementation supports E2HC, the value of SMMU_IDR2.E2HS is 1. The register bit assignments for this TYPE are:

No support for E2HC, or SMMU_CR0.HYPMODE == 0



SMMU_CR0.HYPMODE == 1



The E2HC bit is architecturally mapped to the HYPC bit. However, statements in the architecture about HYPC do not apply to E2HC, despite the architectural mapping of these bits.

Register field descriptions, for both values of SMMU_CR0.HYPMODE

IRPTNDX[7:0], bits[31:24]

Interrupt Index. The Context interrupt number to assert in the event of an interrupt raising a fault in the associated translation context bank. SMMU_IDR0.NUMIRPT specifies the range of values that software can configure this field in.

It is IMPLEMENTATION DEFINED whether this field is fully implemented with writable storage. See *Context interrupts on page 3-100* for more information.

In SMMUv2, each context bank has a dedicated interrupt pin, and there is therefore no requirement for a Context interrupt number. See *Context interrupts on page 3-100* for more information.

WACFG, bits[23:22]

Write-Allocate Configuration, allocation hint for write accesses. The encoding of this field is:

`0b00` Default allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Write-Allocate.

`0b11` No Write-Allocate.

In an implementation that does not support Write-Allocate configuration, this field is RAZ/WI.

RACFG, bits[21:20]

Read-Allocate Configuration, allocation hint for read accesses. The encoding of this field is:

`0b00` Default allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Read-Allocate.

`0b11` No Read-Allocate.

In an implementation that does not support Read-Allocate configuration, this field is RAZ/WI.

BSU, bits[19:18]

Barrier Shareability Upgrade.

This field upgrades the required shareability domain of barriers issued by client devices mapped to this Stream mapping register group by setting the minimum shareability domain that is applied to any barrier.

The encoding of this field is:

`0b00` No effect.

`0b01` Inner Shareable.

`0b10` Outer Shareable.

`0b11` Full system.

Upgrade of the barrier shareability domain might not be supported in all system topologies. In an implementation that does not include this upgrade behavior, BSU is RAZ/SBZP.

TYPE, bits[17:16]

Translation context type. See *SMMU_CBARn format on page 10-241* for more information.

MemAttr, bits[15:12]

Memory Attributes. See *Memory attribute, MemAttr on page 9-179*.

FB, bit[11] Force Broadcast.

Force Broadcast of TLB, branch predictor and instruction cache maintenance operations.

This field affects client TLB maintenance, BPIALL and ICIALLU operations. If this field is 1, the affected operation is modified to the equivalent broadcast variant in the Inner Shareable domain.

The possible values of this bit are:

0 Process the affected operations as presented.

1 Upgrade the affected operations to be broadcast in the Inner Shareable domain.

HYPC, bit[10], when HYPC is available

This bit definition applies:

- To any implementation that does not support E2HC. This means to any SMMUv1 implementation, and to an SMMUv2 implementation for which the value of SMMU_IDR2.E2HS is 0.
- To an implementation that supports E2HC, when the value of SMMU_CR0.HYPMODE is 0, making HYPC available.

Hypervisor Context (HYPC) enable. The possible values of this bit are:

0 Context is not HYPC.

1 Context is HYPC.

Note

HYPC corresponds to the EL2 translation regime in the ARM processor architecture when EL2 is not using the Virtualization Host Extensions.

When HYPC is selected, TLB entries allocated for the context bank are tagged as HYPC, with no ASID or VMID tagging.

If this is a Secure context and all of SMMU_CBARn.HYPC, SMMU_CBA2Rn.MONC, and SMMU_CBA2Rn.VA64 are set to 1, then the result of attempting to process a transaction using this context bank is UNPREDICTABLE.

In an implementation that supports two Security states, the following restrictions apply to Secure software:

- If SMMU_SCR1.GASRAE==0, Secure software must not set HYPC to 1 for any Secure translation context bank.
- If SMMU_SCR1.GASRAE==1, Secure software must not set HYPC to 1 for any Non-secure translation context bank.

Otherwise, UNPREDICTABLE behavior might occur.

See *Hypervisor contexts (HYPC) on page 2-86* for more information.

E2HC, bit[10], when E2HC is available

This bit definition applies only to an SMMUv2 implementation for which both:

- E2HC is supported, indicated by an SMMU_IDR2.E2HS value of 1.
- E2HC is available, indicated by an SMMU_CR0.HYPMODE value of 1.

E2H Context (E2HC) enable. The possible values of this bit are:

0 Context is not E2HC.

1 Context is E2HC.

Note

E2HC corresponds to the EL2 translation regime in the ARMv8.1 processor architecture when EL2 is using the Virtualization Host Extensions.

When E2HC is selected, TLB entries allocated for the context bank are tagged as E2HC and, if non-global, are tagged with the ASID.

If this is a Secure context, and all of SMMU_CBARn.E2HC, SMMU_CBA2Rn.MONC, and SMMU_CBA2Rn.VA64 are set to 1, then the result of attempting to process a transaction using this context bank is UNPREDICTABLE.

See *E2H contexts (E2HC) on page 2-90* for more information about behavior when the value of this bit is 1.

BPSHCFG, bits[9:8]

Bypass Shared Configuration. The encoding of this field is:

`0b00` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b01` Outer Shareable.

`0b10` Inner Shareable.

`0b11` Non-shareable.

This field is combined with the shared attributes from the stage 1 translation table walk, as shown in Table 10-2.

Table 10-2 Shared attribute combination results

Stage 1 shareability	Stage 2 shareability	Resulting shareability
Outer Shareable	-	Outer Shareable
Inner Shareable	Outer Shareable	Outer Shareable
Inner Shareable	Inner Shareable	Inner Shareable
Inner Shareable	Non-shareable	Inner Shareable
Non-shareable	Outer Shareable	Outer Shareable
Non-shareable	Outer Shareable	Outer Shareable
Non-shareable	Inner Shareable	Inner Shareable
Non-shareable	Non-shareable	Non-shareable

In an implementation that does not support Bypass Shared configuration, this field is RAZ/WI.

VMID, bits[7:0]

When the value of SMMU_CR0.VMID16EN is 0, indicating the use of 8-bit VMIDs, this field holds the Virtual Machine Identifier (VMID) to be associated with the translation context bank. If SMMU_CR0.VMID16EN is 1, then this field is RES0, and the VMID comes from SMMU_CBA2Rn.VMID16. See *VMID size on page 1-39*.

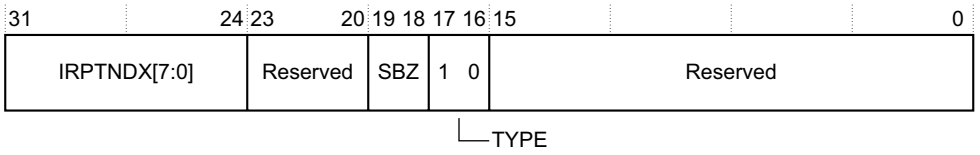
If an implementation that supports two Security states does not support a VMID for Secure translation regimes, any Secure VMID field is ignored. See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

ARM recommends that a Guest OS uses a single fixed VMID value of zero.

Stage 1 context with stage 2 fault, TYPE==0b10

This format configures the associated translation context bank to provide stage 1 translation, additionally specifying the fault context instead of a stage 2 translation context. Any transaction that maps to the corresponding translation context bank incurs an invalid context fault.

The format of the bit assignments is:



IRPTNDX[7:0], bits[31:24]

Interrupt Index. The Context interrupt number to assert in the event of an interrupt raising a fault in the associated translation context bank. SMMU_IDR0.NUMIRPT specifies the range of values that software can configure this field in.

It is IMPLEMENTATION DEFINED whether this field is fully implemented with writable storage. See *Context interrupts on page 3-100* for more information.

In SMMUv2, each context bank has a dedicated interrupt pin, and there is therefore no requirement for a Context interrupt number. See *Context interrupts on page 3-100* for more information.

Bits[23:20] Reserved.

SBZ, bits[19:18]

Should-Be-Zero.

TYPE, bits[17:16]

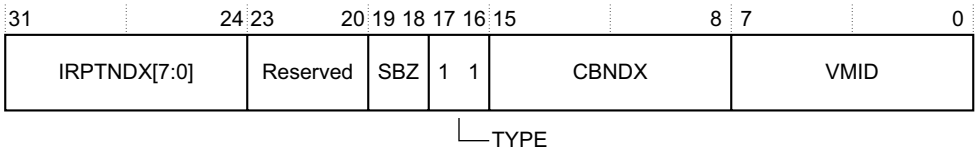
Translation context type. See *SMMU_CBARn format on page 10-241* for more information.

Bits[15:0] Reserved.

Stage 1 followed by stage 2 translation context, TYPE==0b11

This format configures the associated translation context bank to provide stage 1 translation, and specifies an additional translation context bank as the stage 2 translation context bank in a stage 1 followed by stage 2 translation.

The format of the bit assignments is:



IRPTNDX[7:0], bits[31:24]

Interrupt Index. The Context interrupt number to assert in the event of an interrupt raising a fault in the associated translation context bank. SMMU_IDR0.NUMIRPT specifies the range of values that software must configure this field in.

It is IMPLEMENTATION DEFINED whether this field is fully implemented with writable storage. See *Context interrupts on page 3-100* for more information.

In SMMUv2, each context bank has a dedicated interrupt pin, and there is therefore no requirement for a Context interrupt number. See *Context interrupts on page 3-100* for more information.

Bits[23:20] Reserved.

SBZ, bits[19:18] Should-Be-Zero.

TYPE, bits[17:16]

Translation context type. See *SMMU_CBARn format on page 10-241* for more information.

CBNDX, bits[15:8]

Context Bank Index. The translation context bank index used for the stage 2 translation context bank in a stage 1 followed by stage 2 translation.

Behavior is UNPREDICTABLE if the SMMU_CBAR*n* register associated with the translation context specified by SMMU_CBAR*n*.CBNDX has any value other than `0b00` to specify a stage 2 translation context bank.

It is IMPLEMENTATION DEFINED whether this field is fully implemented with writable storage. A portion of this field is permitted to behave as RAZ/WI, because the implementation might not support a sufficient number of translation context banks for some of the upper bits of SMMU_CBAR*n*.CBNDX to be relevant. The implemented range of this field is the same for all SMMU_CBAR*n* registers in an implementation.

VMID, bits[7:0]

When the value of SMMU_CR0.VMID16EN is 0, indicating the use of 8-bit VMIDs, this field holds the Virtual Machine Identifier (VMID) to be associated with the translation context bank. If SMMU_CR0.VMID16EN is set to 1, then this field is RES0, and the VMID comes from SMMU_CBA2Rn.VMID16. See *VMID size on page 1-39*.

Having the VMID of this stage 1 context different from the VMID used in the stage 2 context pointed to by the CBNDX field is UNPREDICTABLE.

10.2.2 SMMU_CBA2R*n*, Context Bank Attribute Registers

The SMMU_CBA2R*n* characteristics are:

Purpose Extends the configuration attributes for the translation context bank that SMMU_CBARn specifies. This register provides support for configuring a context bank to use the AArch64 translation scheme, and as Monitor context.

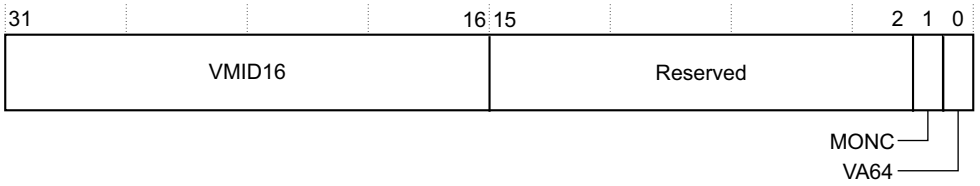
Usage constraints SMMU_IDR1.NUMS2CB specifies the IMPLEMENTATION DEFINED number of translation context banks that only support stage 2 translation. See *The translation context bank table on page 2-75* for more information about discovering whether an implementation uses such context banks.

If a context bank has a stalled transaction, indicated by SMMU_CBn_FSR.SS==1, when the corresponding SMMU_CBA2Rn register is written, an implementation might choose to retry the stalled transaction immediately after SMMU_CBA2Rn is updated. That is, the implementation might exhibit behavior corresponding to SMMU_CBn_RESUME.TnR==0.

Configurations The number of SMMU_CBA2R*n* registers is IMPLEMENTATION DEFINED.

Attributes 32-bit RW registers. See also Table 10-1 on page 10-240.

The SMMU_CBA2Rn bit assignments are:



VMID16, bits[31:16]

16-bit VMID.

This field is RES0 when SMMU_CR0.VMID16EN is 0, or when 16-bit VMIDs are not supported, that is SMMU_CR0.VMID16S == 0.

See *VMID size on page 1-39*.

Note

16-bit VMIDs can be used regardless of the value of SMMU_CBA2Rn.VA64.

In SMMUv1, this field is reserved.

Bits[15:2] Reserved.

MONC, bit[1]

Monitor context bank. The possible values of this bit are:

0 Non-monitor context. Use VMID or ASID for TLB tagging.

1 Monitor context. Do not use VMID or ASID for TLB tagging.

This bit is ignored when any of the following conditions exist:

- SMMU_CBA2Rn.VA64 == 0.
- SMMU_CBARn.TYPE != `0b01`.
- The bank is Non-secure.

If a Secure bank has SMMU_CBARn.HYPC == 1 and SMMU_CBA2Rn.MONC == 1, then the result is UNPREDICTABLE.

In implementations that do not support two Security states, this bit is RAZ/WI.

VA64, bit[0]

Virtual address width. The possible values of this bit are:

0 32-bit. V7S or V7L translation table formats can be used.

1 64-bit. The context bank uses the V8L translation table format.

The SMMU_CBn_TCR.EAE bit selects the format of the 32-bit context bank. This bit is ignored for 32-bit HYPC banks.

Support is provided for stage 2 translation context banks using only the AArch32 Long-descriptor translation scheme.

In an implementation that only supports AArch64 format, this bit may be RAO.

If AArch32 address translation is supported, then this bit must reset to 0.

Changing the Security state of the corresponding context bank makes this bit UNKNOWN.

Note

This means that SMMUv1 Secure software might require updating to avoid a possible security violation on the change of Security state. See *SMMUv2 backwards compatibility problem with SMMUv1 on page 1-20*.

10.2.3 SMMU_CBFRSYNRA*n*, Context Bank Fault Restricted Syndrome Register A

The SMMU_CBFRSYNRA*n* characteristics are:

Purpose Gives fault syndrome information about the access that caused an exception in the associated translation context bank.

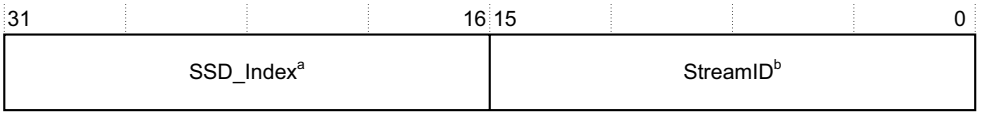
Usage constraints The value of this register is UNKNOWN if the recorded fault was an address translation fault, that is, if the SMMU_CB*n*_FSYNR0.ATOF bit is set to 1.

Configurations The number of SMMU_CBFRSYNRA registers is IMPLEMENTATION DEFINED. See also SMMU_IDR1.NUMCB.

For more information about SMMU_CBFRSYNRA*n*. See *Context Bank Fault Restricted Syndrome Register, SMMU_CBFRSYNRAn on page 3-106*.

Attributes 32-bit RW registers. See also Table 10-1 on page 10-240.

The SMMU_CBFRSYNRA*n* bit assignments are:



a. In SMMUv1, bit[31] is reserved and bits[30:16] are SSD_Index.
b. In SMMUv1, bit[15] is reserved and bits[14:0] are StreamID.

SSD_Index, bits[31:16]

The SSD_Index of the transaction that caused the fault.

The number of SSD_Index bits is IMPLEMENTATION DEFINED. Unimplemented bits behave as RAZ/WI.

This field is only accessible to configuration accesses by Secure software. Non-secure configuration accesses treat this field as RAZ/WI.

Note

When SMMU_SSDR registers are not provided in the SMMU address space, the SSD_Index field is UNKNOWN. See *TLB operation on page 2-72* for more information about the SMMU_SSDR registers.

SMMUv1 does not support 16-bit StreamIDs, and bit[31] is reserved.

StreamID, bits[15:0]

The StreamID of the transaction that caused the fault.

The number of StreamID bits is IMPLEMENTATION DEFINED. Unimplemented bits behave as RAZ/WI.

SMMUv1 does not support 16-bit StreamIDs, and bit[15] is reserved.

Chapter 11
SMMU IMPLEMENTATION DEFINED Address Space

This chapter specifies the SMMU IMPLEMENTATION DEFINED address space. It contains the following section:

- *About the SMMU IMPLEMENTATION DEFINED address space on page 11-252*.

11.1 About the SMMU IMPLEMENTATION DEFINED address space

The SMMU IMPLEMENTATION DEFINED address space is reserved for IMPLEMENTATION DEFINED purposes.

The SMMU architecture supports atomic register accesses at the size of the register. Whether atomic accesses of other sizes are supported is IMPLEMENTATION DEFINED.

The size of this address space is defined by PAGESIZE. See *PAGESIZE and NUMPAGENDXB on page 8-164*.

The content of this address space is relative to the offset from address SMMU_GID_BASE, as Table 11-1 shows.

Table 11-1 SMMU IMPLEMENTATION DEFINED address space

Offset	Name	Description
0x00000-(PAGESIZE − 0x4)	IMPLEMENTATION DEFINED	Reserved for IMPLEMENTATION DEFINED purposes

Chapter 12
SMMU Performance Monitors Extension Register Map

This chapter describes the recommended memory-mapped and external debug interface to the Performance Monitors Extension. It contains the following sections:

12.1 SMMU Performance Monitors Extension register summary

A memory-mapped interface is available for software running on any processor in the system to access counters in the Performance Monitors Extension.

The SMMU architecture supports 32-bit atomic access to the registers in this address space. Whether any other access size is supported is IMPLEMENTATION DEFINED.

Table 12-1 shows a list of the memory-mapped registers for the Performance Monitors Extension.

Table 12-1 SMMU Performance Monitors Extension register map

Offset	Name	Width	Type	Description
0x00000+*m*	PMEVCNTRn	32-bit	RW	*PMEVCNTRn, Performance Monitors Event Count Registers on page 12-265*. *m* is 4 times the event counter number.
(0x00000+*m*)-0x003FC	-	-	-	Reserved. *m* is 4 times the number of event counters.
0x00400+*m*	PMEVTYPERn	32-bit	RW	*PMEVTYPERn, Performance Monitors Event Type Registers on page 12-265*. *m* is 4 times the event counter number.
(0x00400+*m*)-0x007FC	-	-	-	Reserved. *m* is 4 times the number of event counters.
0x00800+*m*	PMCGCRn	32-bit	RW[a]	*PMCGCRn, Performance Monitors Counter Group Configuration Registers on page 12-261*. *m* is 4 times the Counter group number.
(0x00800+*m*)-0x009FC	-	-	-	Reserved. *m* is 4 times the number of Counter groups.
0x00A00+*m*	PMCGSMRn	32-bit	RW	*PMCGSMRn, Performance Monitors Counter Group Stream Match Registers on page 12-262*. *m* is 4 times the Counter group number.
(0x00A00+*m*) -0x00BFC	-	-	-	Reserved. *m* is 4 times the number of Counter groups.
0x00C00-0x00C1C	PMCNTENSETx	32-bit	RW	*PMCNTENSETx, Performance Monitors Count Enable Set registers on page 12-263*.
0x00C20-0x00C3C	PMCNTENCLRx	32-bit	RW	*PMCNTENCLRx, Performance Monitors Counter Enable Clear registers on page 12-262*.
0x00C40-0x00C5C	PMINTENSETx	32-bit	RW	*PMINTENSETx, Performance Monitors Interrupt Enable Set registers on page 12-267*.
0x00C60-0x00C7C	PMINTENCLRx	32-bit	RW	*PMINTENCLRx, Performance Monitors Interrupt Enable Clear registers on page 12-266*.
0x00C80-0x00C9C	PMOVSCLRx	32-bit	RW	*PMOVSCLRx, Performance Monitors Overflow Status Clear registers on page 12-268*.
0x00CA0-0x00CBC	-	-	-	Reserved.
0x00CC0-0x00CDC	PMOVSSETx	32-bit	RW	*PMOVSSETx, Performance Monitors Overflow Status Set registers on page 12-269*.
0x00CE0-0x00D7C	-	-	-	Reserved.
0x00D80-0x00DFC	-	-	-	IMPLEMENTATION DEFINED.
0x00E00	PMCFGR	32-bit	RO[a]	*PMCFGR, Performance Monitors Configuration Register on page 12-260*.
0x00E04	PMCR	32-bit	RW	*PMCR, Performance Monitors Control Register on page 12-264*.
0x00E08	-	-	-	Reserved.

Table 12-1 SMMU Performance Monitors Extension register map (continued)

Offset	Name	Width	Type	Description
0x00E0C-0x00E1C	-	-	-	Reserved.
0x00E20	PMCEID0	32-bit	RO	*PMCEID0, Performance Monitors Common Event Identifier 0 register on page 12-259*, *PMCEID1, Performance Monitors Common Event Identifier 1 register on page 12-260*.
0x00E24	PMCEID1			
0x00E28-0x00E7C	-	-	-	Reserved.
0x00E80-0x00EFC	IMPLEMENTATION DEFINED	-	-	-
0x00F00	-	-	-	Reserved for integration mode control register.
0x00F04-0x00FB4	-	-	-	Reserved.
0x00FB8	PMAUTHSTATUS	32-bit	RO	*PMAUTHSTATUS, Performance Monitors Authentication Status register on page 12-257*.
0x00FBC-0x00FC8	-	-	-	Reserved.
0x00FCC	PMDEVTYPE	32-bit	RO	*PMDEVTYPE, Performance Monitors Device Type Register on page 12-265*.
0x00FD0-0x00FFC	PMPIDy, PMCIDz	32-bit	RO	Performance Monitors Peripheral Identification and Component Identification Registers.
0x01000-(PAGESIZE–0x4)	Reserved	-	-	-

a. See individual field descriptions for variations.

12.2 SMMU Performance Monitors Extension register descriptions

This section describes the SMMU Performance Monitors Extension registers that might be present in an SMMU implementation. Registers are shown in register name order.

The following applies to some of the registers described in this section:

- PMCFGR.N indicates the number of event counters, where:
  - Counters are numbered continuously from 0 to PMCFGR.N.
  - The number of implemented instances of the specified register is (PMCFGR.N DIV 32) +1.
- For performance monitor counter *i*, use:
  - Register *x*, where *x* = *i* DIV 32.
  - Register bit *j*, where *j* = *i* MOD 32.

  For register *x*, *j* takes values from 0 to (*m*-1), where *m* is defined by the following pseudocode:

```
if (x == PMCFGR.N DIV 32) then
    m = (PMCFGR.N MOD 32) + 1;
else if (x ≥ PMCFGR.N DIV 32) then
    m = 0;  // the register is RAZ/WI
else
    x = 32; // all bits are RW.
```

The affected registers are:

- PMCNTENSETx.
- PMCNTENCLRx.
- PMINTENSETx.
- PMINTENCLRx.
- PMOVSCLRx.
- PMOVSSETx.

12.2.1 PMAUTHSTATUS, Performance Monitors Authentication Status register

The PMAUTHSTATUS characteristics are:

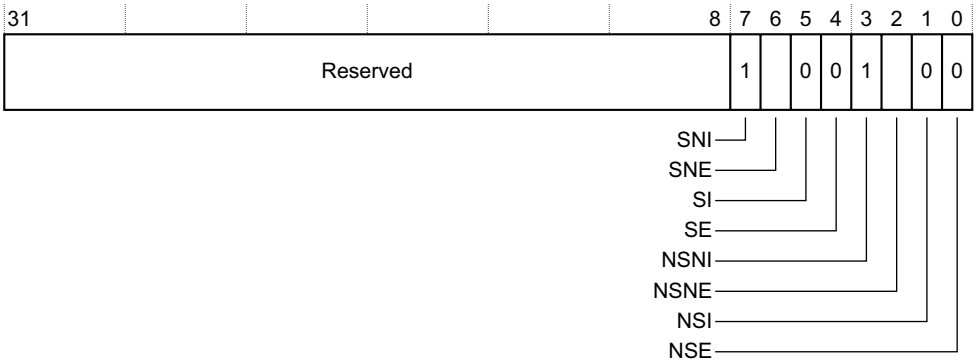
Purpose Indicates the implemented debug features and provides the current values of the configuration inputs that determine the debug permissions.

Usage constraints There are no usage constraints.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RO register.

In an implementation that supports two Security states, the PMAUTHSTATUS bit assignments are:



Bits[31:8] Reserved.

SNI, bit[7] Secure non-invasive debug features implemented. This bit is RAO, Secure non-invasive debug features are implemented.

SNE, bit[6] Secure non-invasive debug enabled. This bit indicates whether counting of Secure transactions is permitted. For the recommended external debug interface, this bit is the logical result of (**DBGEN** OR **NIDEN**) AND (**SPIDEN** OR **SPNIDEN**).

SI, bit[5] Secure invasive debug features implemented. This bit is RAZ, Secure invasive debug features are not implemented.

SE, bit[4] Secure invasive debug enabled. This bit is RAZ.

NSNI, bit[3] Non-secure non-invasive debug features implemented. This bit is RAO, Non-secure non-invasive debug features are implemented.

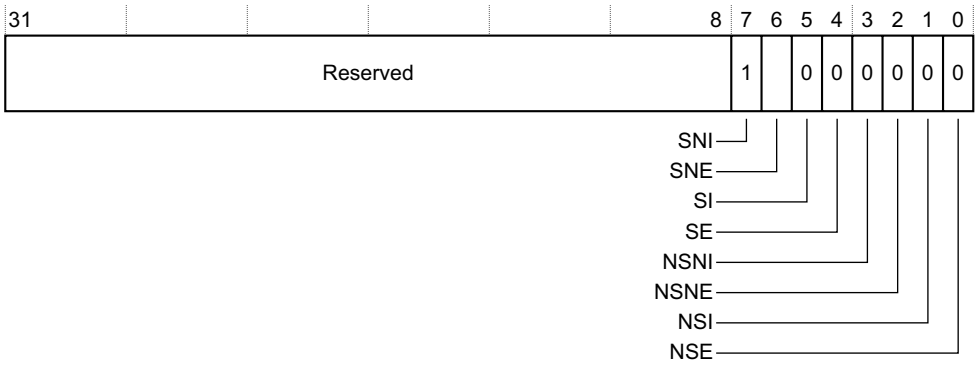
NSNE, bit[2] Non-secure non-invasive debug enabled. For the recommended external debug interface, this bit indicates the logical result of **DBGEN** OR **NIDEN**.

NSI, bit[1] Non-secure invasive debug features implemented. This bit is RAZ, Non-secure invasive debug features are not implemented.

NSE, bit[0] Non-secure invasive debug enabled. This bit is RAZ.

See *About the SMMU Performance Monitors Extension on page 6-146* for more information.

In an implementation that does not support two Security states, the PMAUTHSTATUS bit assignments are:



Bits[31:8] Reserved.

SNI, bit[7] Secure non-invasive debug features implemented. This bit is RAO, Secure non-invasive debug features are implemented.

SNE, bit[6] Secure non-invasive debug enabled. This bit indicates whether non-invasive debug is permitted in Secure PL1 modes. For the recommended external debug interface, this bit is the logical result of **DBGEN** OR **NIDEN**.

SI, bit[5] Secure invasive debug features implemented. This bit is RAZ, Secure invasive debug features are not implemented.

SE, bit[4] Secure invasive debug enabled. This bit is RAZ. It indicates whether invasive halting debug is permitted in Secure PL1 modes.

NSNI, bit[3] Non-secure non-invasive debug features implemented. This bit is RAZ, Non-secure non-invasive debug features are not implemented.

NSNE, bit[2] Non-secure non-invasive debug enabled. This bit is RAZ, Non-secure non-invasive debug is not enabled.

NSI, bit[1] Non-secure invasive debug features implemented. This bit is RAZ, Non-secure invasive debug features are not implemented.

NSE, bit[0] Non-secure invasive debug enabled. This bit is RAZ, Non-secure invasive debug is not enabled.

12.2.2 PMCEID0, Performance Monitors Common Event Identifier 0 register

The PMCEID0 register characteristics are:

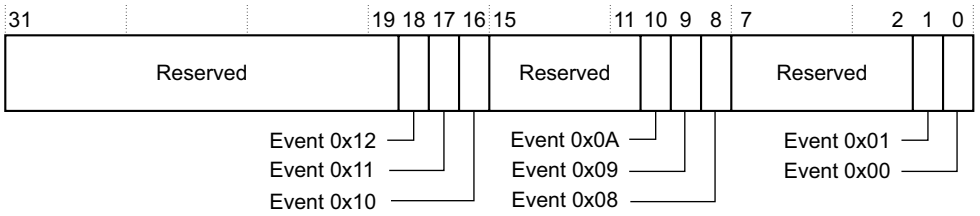
Purpose Describes the event classes supported by the SMMU implementation.

Usage constraints There are no usage constraints.

Configurations See *About the SMMU Performance Monitors Extension on page 6-146* for information about when this register is configured.

Attributes A 32-bit RO register.

The PMCEID0 bit assignments are:



Bits[31:19] Reserved.

Event 0x12, bit[18]

The possible values of this bit are:

0 Access write event not implemented.

1 Access write event implemented.

Event 0x11, bit[17]

The possible values of this bit are:

0 Access read event not implemented.

1 Access read event implemented.

Event 0x10, bit[16]

The possible values of this bit are:

0 Access event not implemented.

1 Access event implemented.

Bits[15:11] Reserved.

Event 0x0A, bit[10]

The possible values of this bit are:

0 TLB entry allocated for write event not implemented.

1 TLB entry allocated for write event implemented.

Event 0x09, bit[9]

The possible values of this bit are:

0 TLB entry allocated for read event not implemented.

1 TLB entry allocated for read event implemented.

Event 0x08, bit[8]

The possible values of this bit are:

0 TLB entry allocated for event not implemented.

1 TLB entry allocated for event implemented.

Bits[7:2] Reserved.

Event 0x01, bit[1]

The possible values of this bit are:

0 Cycle count divided by 64 event not implemented.

1 Cycle count divided by 64 event implemented.

Event 0x00, bit[0]

The possible values of this bit are:

0 Cycle count event not implemented.

1 Cycle count event implemented.

12.2.3 PMCEID1, Performance Monitors Common Event Identifier 1 register

Whether an SMMU supports the Performance Monitors Extension is IMPLEMENTATION DEFINED:

- If this extension is supported, all of the bits in this 32-bit read-only register are reserved and are UNK.
- If this extension is not supported, the register map corresponding to the Performance Monitors registers is UNK/SBZP.

12.2.4 PMCFGR, Performance Monitors Configuration Register

The PMCFGR characteristics are:

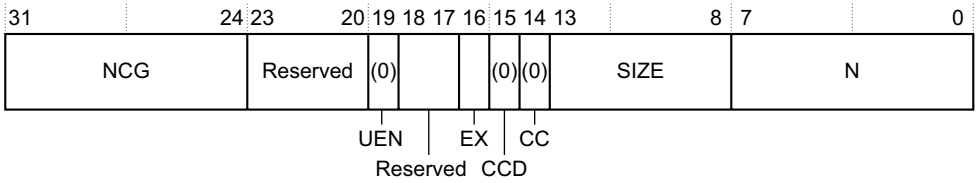
Purpose Provides *Performance Monitoring Unit* (PMU) configuration data.

Usage constraints There are no usage constraints.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RO register.

The PMCFGR bit assignments are:



NCG, bit[31:24] Number of Counter Groups.

The number of Counter groups implemented is calculated as NCG+1. For example, if NCG==`0x00`, there is one implemented Counter group.

Bits[23:20] Reserved.

UEN, bit[19] Unprivileged-mode Enable, reads as the value 0 indicating that this feature is not supported.

Bits[18:17] Reserved.

EX, bit[16] Event export. The possible values of this bit are:

0 Event export is not supported. PMCR.X is RAZ/WI.

1 Event export is supported. PMCR.X is writable.

CCD, bit[15] Cycle Counter pre-scale, reads as the value 0 indicating that there is no cycle counter pre-scale.

CC, bit[14] Cycle Counter, reads as the value 0 indicating that a dedicated cycle counter is not implemented.

SIZE, bits[13:8]

Counter size, reads as the value `0b011111` indicating 32-bit event counters.

N, bits[7:0] Indicates the number of implemented event counters, up to a maximum of 256 counters. The number of counters implemented is N+1.

Note

In the processor PMU, the number of counters implemented is N. This is because the CPU PMU implements an extra counter, PMCCNTR, that is not defined in the SMMU architecture.

12.2.5 PMCGCR*n*, Performance Monitors Counter Group Configuration Registers

The PMCGCR*n* characteristics are:

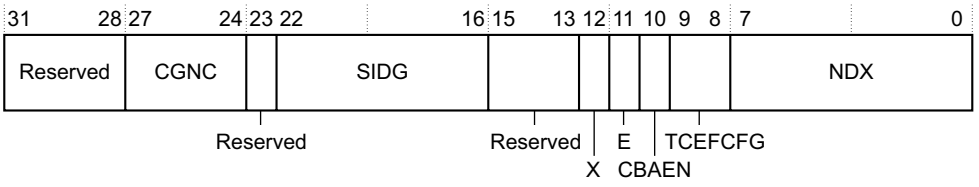
Purpose Controls Counter group behavior.

Usage constraints There are no usage constraints.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMCGCR*n* bit assignments are:



Bits[31:28] Reserved.

CGNC, bits[27:24]

Counter Group Number of Counters, is the number of counters in this Counter group, in the range 1-15, so that a value of `0b0001` == 1 counter.

This field is RO/WI.

Bit[23] Reserved.

SIDG, bits[22:16]

StreamID Group, indicates the StreamID group that this Counter group is affiliated with.

This field is RO/WI.

Bits[15:13] Reserved.

X, bit[12] Export, corresponds to the Performance Monitors Event Export field, SMMU_CBn_PMCR.X, for this Counter group.

E, bit[11] Count Enable, corresponds to the Performance Monitors Count Enable field, SMMU_CBn_PMCR.E, for this Counter group.

CBAEN, bit[10]

Context Bank Assignment Enable.

The possible values of this bit are:

0 Do not reveal Counter group *n* in the translation context bank specified by PMCGCR*n*.NDX.

1 Reveal Counter group *n* in the translation context bank specified by PMCGCR*n*.NDX.

If PMCGCR*n*.CBAEN==1 and PMCGCR*n*.TCEFCFG != (`0b10` or `0b01`), behavior is UNPREDICTABLE.

TCEFCFG, bits[9:8]

Translation Context Event Filtering Configuration.

The possible values of this bit are:

`0b00` Count events on a global basis.

`0b01` Counter events are restricted to matches in the corresponding PMCGSMRn register.

`0b10` Counter events are restricted to the translation context bank indicated by PMCGCR*n*.NDX.

`0b11` Reserved.

NDX, bits[7:0] Index, only relevant when PMCGCRn.CBAEN==0b1 or PMCGCRn.TCEFCFG==0b10. Otherwise, PMCGCR*n*.NDX is SBZ.

12.2.6 PMCGSMR*n*, Performance Monitors Counter Group Stream Match Registers

The PMCGSMR*n* characteristics are:

Purpose Specifies StreamID filtering of the events counted in a Counter group.

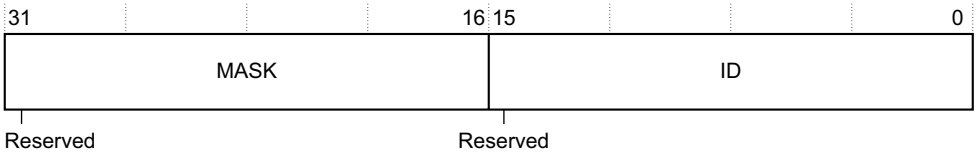
For information about enabling StreamID event filtering, see the corresponding PMCGCRn.TCEFCFG field.

Usage constraints There are no usage constraints.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMCGSMR*n* bit assignments are:



MASK, bits[31:16]

If MASK[i] == 1, ID[i] is ignored and SBZ.

If MASK[i] == 0, ID[i] is valid for matching.

The number of MASK bits actually present is IMPLEMENTATION DEFINED. Unimplemented bits are RAZ/WI.

If SMMU_sCR0.EXIDENABLE is 0, bit[31] is reserved.

ID, bits[15:0] The StreamID to match.

The number of ID bits actually present is IMPLEMENTATION DEFINED. Unimplemented bits are RAZ/WI.

If SMMU_sCR0.EXIDENABLE is 0, bit[15] is reserved.

12.2.7 PMCNTENCLR*x*, Performance Monitors Counter Enable Clear registers

The PMCNTENCLR*x* registers characteristics are:

Purpose Disables any implemented event counter, PMN*i*, for *i* in the range ($x \times 32$) to ($x \times 32$) + 31.

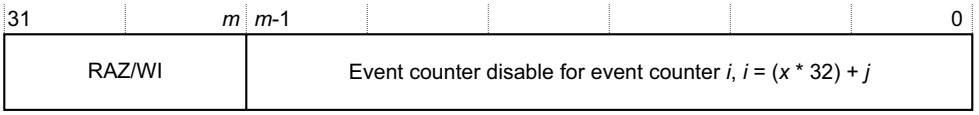
Reading this register indicates the counters that are enabled.

Usage constraints Used in conjunction with PMCNTENSETx.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMCNTENCLR*x* register bit assignments are:



Note

See *SMMU Performance Monitors Extension register descriptions on page 12-256* for the definitions of *m*, *x*, and *j*.

Bits[31:*m*] RAZ/WI.

P*i*, bit[*j*], for *j* = 0 to (*m*–1)

Event counter *i* disable bit, where $i = (x \times 32) + j$.

Table B4-3 shows the behavior of this bit on reads and writes.

Table 12-2 Read and write values for the PMCNTENCLR*x*.P*i* bits

P*x* value	Meaning on read	Action on write
0	PMN*i* event counter disabled	No action, write is ignored
1	PMN*i* event counter enabled	Disable the PMN*i* event counter

Note

PMCR.E can override the settings in this register and disable all counters. PMCNTENCLR*x* retains its value when PMCR.E is 0, even though its settings are ignored.

12.2.8 PMCNTENSET*x*, Performance Monitors Count Enable Set registers

The PMCNTENSET*x* registers characteristics are:

Purpose Enables any implemented event counters, PMN*i*, for *i* in the range $(x \times 32)$ to $(x \times 32) + 31$.

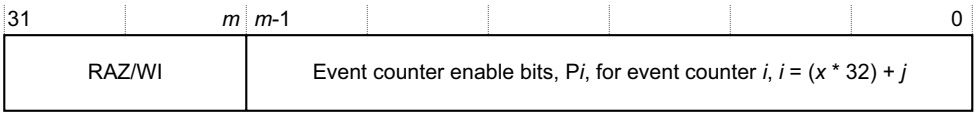
Reading a PMCNTENSET*x* register indicates the counters that are enabled.

Usage constraints Use in conjunction with PMCNTENCLRx.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMCNTENSET*x* register bit assignments are:



Note

See *SMMU Performance Monitors Extension register descriptions on page 12-256* for the definitions of *m*, *x* and *j*.

Bits[31:*m*] RAZ/WI.

P*i*, bit[*j*], for *j* = 0 to (*m*–1)

Event counter *i* enable bit, where $i = (x \times 32) + j$.

Table B4-5 shows the behavior of this bit on reads and writes.

Table 12-3 Read and write values for the PMCNTENSET*x*.P*i* bits

P*x* value	Meaning on read	Action on write
0	PMN*i* event counter disabled.	No action, write is ignored.
1	PMN*i* event counter enabled.	Enable the PMN*i* event counter.

12.2.9 PMCR, Performance Monitors Control Register

The PMCR characteristics are:

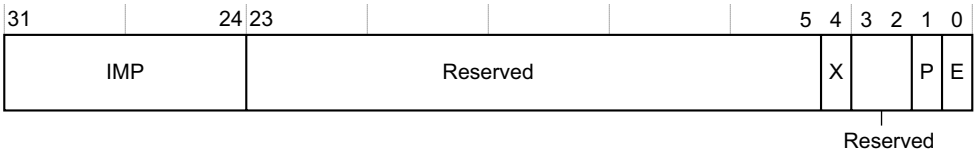
Purpose PMCR provides controls for the Performance Monitors.

Usage constraints There are no usage constraints.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMCR bit assignments are:



IMP, bits[31:24] Implementor code that is either:
- Allocated by ARM.
- The value 0.

This RO field is IMPLEMENTATION DEFINED.

Bits[23:5] Reserved.

X, bit[4] Export enable. The possible values of this bit are:

0 Export of events is disabled.

1 Export of events is enabled.

See *SMMU_CBn_PMCR banking on page 6-152* for more information.

Bits[3:2] Reserved.

P, bit[1] Event counter reset. This bit is WO. The effects of writing to this bit are:

0 No action.

1 Reset all event counters to zero.

See *SMMU_CBn_PMCR banking on page 6-152* for more information.

E, bit[0] Enable. The possible values of this bit are:

0 All counters, including PMCCNTR, are disabled.

1 All counters are enabled.

Overflow interrupts are only enabled if the event counters are enabled.

See *SMMU_CBn_PMCR banking on page 6-152* for more information.

12.2.10 PMDEVTYPE, Performance Monitors Device Type Register

The PMDEVTYPE characteristics are:

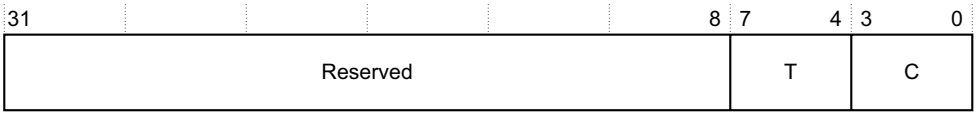
Purpose Provides the CoreSight device type information for the Performance Monitors, and indicates the type of debug component.

Usage constraints There are no usage constraints.

Configurations Must be implemented in all CoreSight components.

Attributes A 32-bit RO register.

The PMDEVTYPE bit assignments are:



Bits[31:0] Reserved.

T, bits[7:4] Sub-type, a fixed value of `0x5`, indicating association with a memory management unit conforming to the ARM SMMU architecture.

C, bits[3:0] Class, a fixed value of `0x6`, indicating a Performance Monitor device type.

12.2.11 PMEVCNTR*n*, Performance Monitors Event Count Registers

The PMEVCNTR*n* characteristics are:

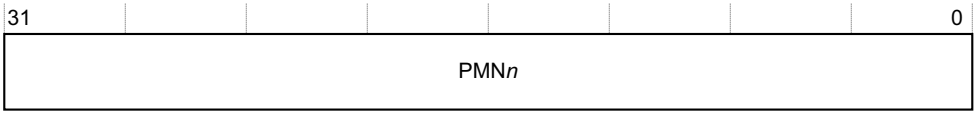
Purpose Reads or writes the value of the selected event counter, PMN*n*.

Usage constraints There are no usage constraints.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMEVCNTR*n* bit assignments are:



PMNx, bits[31:0] The value of the selected event counter, PMN*n*.

Note

PMEVCNTR*n* can be written to by software even when the counter is disabled. This is true regardless of whether the counter is disabled because either:

- A 1 has been written to the appropriate bit in PMCNTENCLRx.
- PMCR.E is 0.

12.2.12 PMEVTYPER*n*, Performance Monitors Event Type Registers

The PMEVTYPER*n* characteristics are:

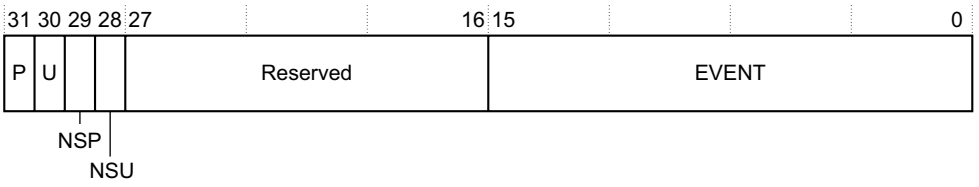
Purpose Controls which events are counted by the corresponding event counter.

Usage constraints There are no usage constraints.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMEVTYPER*n* bit assignments are:



P, bit[31] Privileged transactions filtering bit. Controls the counting of Secure privileged transactions.

The possible values of this bit are:

0 Count events relating to Secure privileged transactions.

1 Do not count events relating to Secure privileged transactions.

U, bit[30] Unprivileged transactions filtering bit. Controls the counting of Secure unprivileged transactions.

The possible values of this bit are:

0 Count events relating to Secure unprivileged transactions.

1 Do not count events relating to Secure unprivileged transactions.

NSP, bit[29] Non-secure Privileged transactions filtering bit. Controls the counting of Non-secure privileged transactions.

The behavior depends on the combined values of the P and NSP bits. The possible values of this bit are:

P==NSP Count events relating to Non-secure privileged transactions.

P!=NSP Do not count events relating to Non-secure privileged transactions.

NSU, bit[28] Non-secure unprivileged transactions filtering bit. Controls counting of Non-secure unprivileged transactions.

The behavior depends on the combined values of the U and NSU bits. The possible values of this bit are:

U==NSU Count events relating to Non-secure unprivileged transactions.

U!=NSU Do not count events relating to Non-secure unprivileged transactions.

Bits[27:16] Reserved.

EVENT, bits[15:0]

Event type. See *Event classes on page 6-148*.

It is IMPLEMENTATION DEFINED whether bits[15:8] are RAZ/WI or implemented, depending on the number of events that the implementation supports.

12.2.13 PMINTENCLR*x*, Performance Monitors Interrupt Enable Clear registers

The PMINTENCLR*x* registers characteristics are:

Purpose Disables the generation of interrupt requests on overflows from each implemented event counter, PMN*i*, for *i* in the range ($x \times 32$) to ($x \times 32$) + 31.

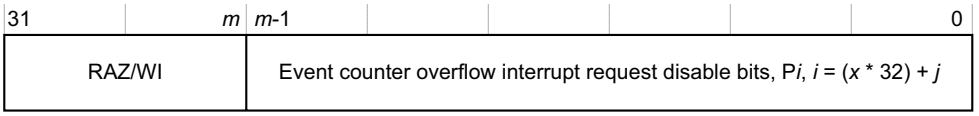
Reading the register indicates the overflow interrupt requests that are enabled.

Usage constraints Used in conjunction with the PMINTENSETx register.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMINTENCLR*x* register bit assignments are:



Note

See *SMMU Performance Monitors Extension register descriptions on page 12-256* for the definitions of *m*, *x*, and *j*.

Bits[31:*m*] RAZ/WI.

P*i*, bit[*j*], for *j* = 0 to (*m*–1)

Event counter *i*, overflow interrupt request disable bit, where $i = (x \times 32) + j$.

Table B4-7 shows the behavior of this bit on reads and writes.

Table 12-4 Read and write values for the PMINTENCLR*x*.P*i* bits

P*i* value	Meaning on read	Action on write
0	PMN*i* interrupt request disabled.	No action, write is ignored.
1	PMN*i* interrupt request enabled.	Disable the PMN*i* interrupt request.

12.2.14 PMINTENSET*x*, Performance Monitors Interrupt Enable Set registers

The PMINTENSET*x* registers characteristics are:

Purpose Enables the generation of interrupt requests on overflows from each implemented event counter, PMN*i*, for *i* in the range $(x \times 32)$ to $(x \times 32) + 31$.

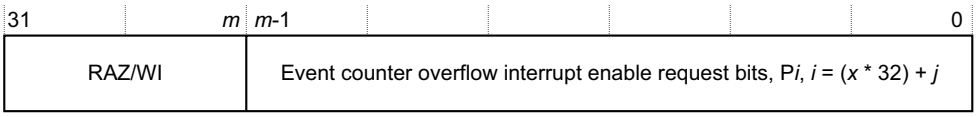
Reading PMINTENSET*x* indicates the overflow interrupt requests that are enabled.

Usage constraints Used in conjunction with PMINTENCLRx.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMINTENSET*x* register bit assignments are:



Note

See *SMMU Performance Monitors Extension register descriptions on page 12-256* for the definitions of *m*, *x*, and *j*.

Bits[31:*m*] RAZ/WI.

P*i*, bit[*j*], for *j* = 0 to (*m*–1)

Event counter *i*, overflow interrupt request enable bit, where $i = (x \times 32) + j$.

Table B4-9 shows the behavior of this bit on reads and writes.

Table 12-5 Read and write values for the PMINTENSET*x*.P*i* bits

P*i* value	Meaning on read	Action on write
0	PMN*i* interrupt request disabled.	No action, write is ignored.
1	PMN*i* interrupt request enabled.	Enable the PMN*i* interrupt request.

When an interrupt is signaled, software can remove it by writing a 1 to the corresponding overflow bit in PMOVSCLRx.

Note

ARM expects that the interrupt request that can be generated on a counter overflow is exported from the processor, meaning it can be factored into a system interrupt controller if applicable. This means that normally the system has more levels of control of the interrupt generated.

12.2.15 PMOVSCLR*x*, Performance Monitors Overflow Status Clear registers

The PMOVSCLR*x* registers characteristics are:

Purpose Clears the state of the overflow bit for each implemented event counter, PMN*i*, for *i* in the range $(x \times 32)$ to $(x \times 32) + 31$.

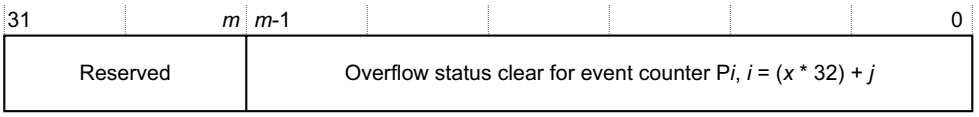
Reading the register shows the current overflow status.

Usage constraints There are no usage constraints.

Configurations Implemented only as part of the Performance Monitors Extension.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMOVSCLR*x* bit assignments are:



Note

See *SMMU Performance Monitors Extension register descriptions on page 12-256* for the definitions of *m*, *x*, and *j*.

Bits[31:*m*] RAZ/WI.

P*i*, bit[*j*], for *j* = 0 to (*m*–1)

Event counter *i*, overflow status clear bit, where $i = (x \times 32) + j$.

Table B4-7 shows the behavior of this bit on reads and writes.

Table 12-6 Read and write values for the PMOVSCLR*x*.P*i* bits

P*i* value	Meaning on read	Action on write
0	Counter PMN*i* has not overflowed.	No action, write is ignored.
1	Counter PMN*i* has overflowed.	Set bit to 0.

12.2.16 PMOVSSET*x*, Performance Monitors Overflow Status Set registers

The PMOVSSET*x* registers characteristics are:

Purpose Sets the state of the overflow bit for each of the implemented event counters, PMN*i*, for *i* in the range ($x \times 32$) to ($x \times 32$) + 31.

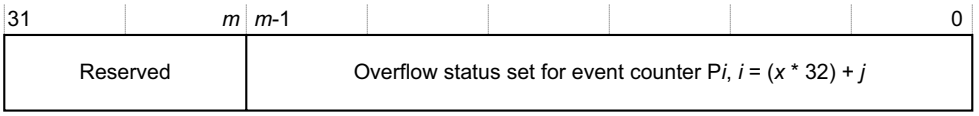
Reading the register shows the current overflow status.

Usage constraints Implemented only as part of the Performance Monitors Extension.

Configurations See *About the SMMU Performance Monitors Extension on page 6-146* for information about when this register is configured.

Attributes A 32-bit RW register with an UNKNOWN reset value.

The PMOVSSET*x* bit assignments are:



Note

See *SMMU Performance Monitors Extension register descriptions on page 12-256* for the definitions of *m*, *x*, and *j*.

Bits[31:*m*] RAZ/WI.

P*i*, bit[*j*], for *j* = 0 to (*m*–1)

Event counter *i*, overflow status set bit, where $i = (x \times 32) + j$.

Table B4-7 shows the behavior of this bit on reads and writes.

Table 12-7 Read and write values for the PMINTENCLR*x*.P*i* bits

P*i* value	Meaning on read	Action on write
0	Counter PMN*i* has not overflowed.	No action, write is ignored.
1	Counter PMN*i* has overflowed.	Set bit to 1.

12.2.17 PMPID*y*, PMCID*z*, Performance Monitors Peripheral ID and Component ID registers

The Peripheral ID and Component ID registers, PMPID0-PMPID7 and PMCID0-PMCID3, are RO registers that provide standard CoreSight peripheral and component identification. For details, see the definitions of DBGPID*n* and DBGCID*n* in the *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

Note

The part number defined by PMPID1[3:0] and PMPID0[7:0] must be a different part number to that defined by DBGPID1[3:0] and DBGPID0[7:0].

Chapter 13
The Security State Determination Address Space

This chapter gives an overview of the SSD address space. It contains the following section:

- *SMMU SSD address space on page 13-272*.

13.1 SMMU SSD address space

The SSD address space can indicate whether each SSD_Index is acting for the Secure domain or the Non-secure domain. This address space is only accessible to Secure memory transactions and is UNKNOWN and WI for Non-secure accesses.

The SMMU architecture supports 32-bit atomic access to the SMMU_SSDRn registers. Whether any other access size is supported is IMPLEMENTATION DEFINED.

In SMMUv1, a single page is reserved for the SSD address space, and SMMUv2 contains an additional page.

The address space might not be fully populated, depending on the implemented PAGESIZE and the SSD_Index width. The SSD_Index width can be read from SMMU_IDR1.NUMSSDNDXB:

- In SMMUv1, the SSD_Index width is SMMU_IDR1.NUMSSDNDXB.
- In SMMUv2, when SMMU_IDR1.SSDTP==0b11 the SSD_Index is 16 bits wide, otherwise it is SMMU_IDR1.NUMSSDNDXB bits wide

Unimplemented SSD_Index bits behave as though they are zero. SSD register bits corresponding to values above the implemented SSD_Index size behave as UNK/WI.

One bit is provided for each SSD_Index value. An SMMU implementation supports an SSD_Index of up to:

- 15 bits in size for SMMUv1, meaning that 4KB of storage is required, occupying a complete page.
- 16 bits in size for SMMUv2, meaning that 8KB of storage is required, and:
  - If PAGESIZE is 4KB, SMMU_SSDRn occupies two pages.
  - If PAGESIZE is 64KB, all SMMU_SSDRn fit in a single page and the remainder of the two pages are reserved.

Note

The calculation of the offset of a given SMMU_SSDRn from the start of the SSD address space is independent of the implemented PAGESIZE.

SSD bits can have fixed values that correspond to SSD_Index values having fixed Secure or Non-secure ownership. Software can detect programmable bits by using a read-modify-write sequence.

Whether the SSD registers are implemented is IMPLEMENTATION DEFINED. An SMMU implementation and the system it is integrated in can use alternative means to resolve the Secure status of transactions. SMMU_IDR1.SSDTP indicates the presence of the SSD table. In an implementation where the registers are not present, this address space behaves as UNK/WI.

The SSD address space layout is shown in terms of offsets from *SMMU_SSD_BASE*. See Table 13-1.

Table 13-1 SSD address space

Offset	Name	Description
0x00000	SMMU_SSDR0	Corresponds to SSD_Index values 0-31.
0x00004	SMMU_SSDR1	Corresponds to SSD_Index values 32-63.
0x00008 - 0x00FFC	SMMU_SSDR2 - SMMU_SSDR1023	Corresponds to SSD_Index values 64-32767.
0x01000 - 0x01FFC	SMMU_SSDR1024 - SMMU_SSDR2047	Corresponds to SSD_Index values 32768-65355. Applies to SMMUv2 only.
0x01000 - (PAGESIZE−0x4)	Reserved	4KB PAGESIZE.
0x02000 - (PAGESIZE−0x4)	Reserved	64KB PAGESIZE, SMMUv2 only.

If the SSD register space is implemented, the behavior of each SMMU_SSDR*n* bit is:

```
// SMMU_SSDRn selected using SSD_Index<15:5>
if (SMMU_SSDRn[SSD_Index<4:0>] == 1) {
   // Transaction is Non-secure
} else {
   // Transaction is Secure
}
```

See *Security State Determination (SSD) on page 2-50* for more information.

Chapter 14
Extended Stream Matching Extension

This chapter specifies an optional extension that supports up to 1024 Stream Match Register Groups. It contains the following sections:

14.1 About the Extended Stream Match extension

The Extended Stream Match extension is an optional extension to the SMMUv2 architecture that supports large systems where 128 Stream Match Register groups are insufficient. For example, it can be implemented where a large number of context banks and a large StreamID space is required.

This extension extends the number of Stream Match Register groups to 1024. To distinguish these from the register groups provided when the extension is not implemented, they are referred to as the *Extended Stream Match Register groups*. When the extension is implemented, the 128 Stream Match Register groups, {SMMU_SMRn, SMMU_S2CRn}, map onto the last 128 Extended Stream Match Register groups, {SMMU_EXSMRn, SMMU_EXS2CRn}. This means, if N is the number of implemented Extended Stream Match Register groups, then:

{SMMU_SMRp, SMMU_S2CRp} map onto {SMMU_EXSMRq, SMMU_S2CRq}

Where $q = (N+p)-128$.

SMMU_IDR0.EXSMRGS identifies whether an implementation includes the Extended Stream Match extension. This bit is 1 if the extension is implemented. When the extension is implemented:

- SMMU_sCR2.EXSMRGENABLE enables the use of the Extended Stream Match Register groups. When the value of this bit is 1, use of these groups is enabled. The banking of SMMU_sCR2 by Security state provides control of the use of these registers by Security state:
  - SMMU_SCR2.EXSMRGENABLE enables use of the Extended Stream Match Register groups for Secure transactions.
  - SMMU_CR2.EXSMRGENABLE enables use of the Extended Stream Match Register groups for Non-secure transactions.

  Note

  Throughout this section, *Secure transaction* or *Non-secure transaction* refers to the Security state determined by the SSD process described in *Security State Determination (SSD) on page 2-50*.

- In the Secure copy of SMMU_sCR2, setting SMMU_SCR2.EXNSSMRGDISABLE to 1 disables the use of the Extended Stream Match Register groups by Non-secure contexts.

Using the Extended Stream Match Register groups does not affect the Stream matching process described in *StreamID matching on page 2-57*, except that the Extended Stream Match extension provides a larger set of register groups for this matching process.

Note

- Although Secure software can use SMMU_SCR2.EXNSSMRGDISABLE to control Non-secure use of the Extended Stream Match Register groups, ARM expects this would be done only by specialized software.
- The mapping of the 128 Stream Match register groups onto the Extended Stream Match Register groups means that Non-secure software can use the extended register set even if the Secure software has no awareness of the Extended Stream Matching Extension.
- ARM recommends that the number of Extended Stream Match Register groups is a multiple of 128. This is not an architectural requirement, but following this recommendation simplifies the calculations for aliasing the Stream Match Register groups to the last 128 Extended Stream Match Register groups, and for calculating the Non-secure value of SMMU_IDR2.EXNUMSMRG.
- Secure software can reserve 128 or more stream matching register groups and leave no stream matching register groups for Non-secure software. ARM believes that such a configuration is unlikely to be useful in real applications of SMMUs that support two Security states. However, if Secure software does reserve 128 or more Stream Matching Register groups, it must ensure that SMMU_SCR1.NSNUMSMRGO correctly indicates that no stream matching register groups are available for Non-secure software.

The values of the Extended Stream Match Register groups that are not mapped to {SMMU_SMRn, SMMU_S2CRn} groups are UNKNOWN on reset, and ignore writes if they have not been enabled in the Security state to which they are assigned. Therefore, software that uses these registers must initialize them after enabling them but before they are used for stream matching. See also *Implications of the Extended Stream Match Extension on save and restore of the SMMU*.

When the Extended Stream Matching Extension is implemented a Secure read of SMMU_IDR0.NUMSMRG returns the value 128.

Note

A Non-secure read of SMMU_IDR0.NUMSMRG always returns the value of SMMU_SCR1.NSNUMSMRGO, regardless of whether the Extended Stream Matching Extension is implemented.

14.1.1 The number of Extended Stream Match Register groups available to Non-secure transactions

In an implementation that includes the Extended Stream Matching Extension, a Non-secure read of SMMU_IDR2.EXNUMSMRG returns the number of Extended Stream Match Register groups that can be used by Non-secure transactions. This value depends on how Stream matching is configured for Secure transactions, and in particular on whether Secure transactions are using Extended Stream Matching, as indicated by the value of the EXSMRGENABLE bit in the Secure instance of the SMMU_sCR2 register, SMMU_SCR2.

Note

When the Extended Stream Matching Extension is implemented, the number of Extended Stream Match Register groups must include the 128 groups that the {SMMU_SMRn, SMMU_S2CRn} register groups map onto.

This means the value returned by a Non-secure read of SMMU_IDR2.EXNUMSMRG is determined as follows:

Extended Stream Match Register groups not used by Secure transactions

When the value of SMMU_SCR2.EXSMRGENABLE is 0, use of the Extended Stream Match Register groups by Secure transactions is disabled. This means that Secure transactions use some of the 128 {SMMU_SMRn, SMMU_S2CRn} register groups. SMMU_SCR1.NSNUMSMRGO defines the number of these groups that are available for Non-secure transactions, and therefore the number reserved for Secure transactions is (128-SMMU_SCR1.NSNUMSMRGO).

These register groups map onto the highest-numbered Extended Stream Match Register groups, and therefore if N is the number of implemented Extended Stream Match Register groups, the value returned by a Non-secure read of SMMU_IDR2.EXNUMSMRG is:

N-(128-SMMU_SCR1.NSNUMSMRGO).

Some Extended Stream Match Register groups used by Secure transactions

When the value of SMMU_SCR2.EXSMRGENABLE is 1, use of the Extended Stream Match Register groups by Secure transactions is enabled, and SMMU_SCR2.EXNSNUMSMRGO defines the number of Extended Stream Match Register groups that are reserved for use by Non-secure transactions. This is the value that is returned by a Non-secure read of SMMU_IDR2.EXNUMSMRG.

The Extended Stream Match Register groups used by Non-secure transactions always start from group 0.

14.1.2 Implications of the Extended Stream Match Extension on save and restore of the SMMU

While SMMU_sCR2.EXSMRGENABLE is 0, the extra register groups that are not aliased to SMMU_S2CRn or SMMU_SMRn are RES0, and the SMMU implementation can put these register groups into a low-power state. This transition to a low-power state might be made as soon as software sets SMMU_sCR2.EXSMRGENABLE to 0, resulting in the loss of the data previously held in those registers.

This places restrictions on software save-and-restore code for power-state management. The Extended Stream Match Register groups, for a Security state, must be saved before the SMMU_sCR2.EXSMRGENABLE bit for that Security state is set to 0.

In addition, the Non-secure and Secure software must on whose responsibility it is to save-and-restore these registers. For example, Non-secure software might save the Non-secure set and the Secure software might save the Secure set. This means that if the Secure software is unaware of the extended register set then the Non-secure software can still use them and would save-and-restore those that are under its control as part of the power-down sequence.

14.2 Extended Stream Match Extension registers

The following sections describe the Extended Stream Match Extension registers:

- *SMMU_EXSMRn, Extended Stream Match Register*
- *SMMU_EXS2CRn, Extended Stream-to-Context Register on page 14-280*

These registers are functionally equivalent to the SMMU_SMRn and SMMU_S2CRn registers, and the SMMU_SMRn and SMMU_S2CRn registers map onto the final 128 SMMU_EXSMRn and SMMU_EXS2CRn registers.

The SMMU architecture supports 32-bit atomic register accesses to these registers. Whether atomic accesses of other sizes are supported is IMPLEMENTATION DEFINED.

As with other registers in the Global address space, when the value of SMMU_SCR1.GASRAE is 1, these registers are only accessible by Secure accesses.

14.2.1 SMMU_EXSMR*n*, Extended Stream Match Register

The SMMU_EXSMR*n* characteristics are:

Purpose Matches a transaction with a particular Stream mapping register group.

Usage Constraints During configuration, the Stream Match Register table can have multiple entries that match the same Stream Identifier value, possibly resulting in UNPREDICTABLE behavior. See *StreamID matching on page 2-57* for more information about multiple matches. To prevent multiple matches, software must ensure that no transactions that might match the StreamID are received, by:

- Disabling any source client devices that might match.
- Ensuring that no outstanding transactions from these client devices are in progress.

Note

As an extra precaution, software can first disable all affected SMMU_EXSMR*n* table entries by setting the SMMU_EXSMRn.VALID bit to 0, then reprogramming the entries as appropriate.

For every implementation of Stream Match Register, an implementation must provide:

- The same number of ID bits and MASK bits.
- The same number of EXID bits and EXMASK bits.

It is IMPLEMENTATION DEFINED whether ID or EXID bits are preserved when the corresponding MASK or EXMASK bit is set to 1 to indicate that the corresponding bit of ID or EXID is ignored in matching.

Configurations This register cannot be implemented in SMMUv1,

The number of SMMU_EXSMR*n* registers is IMPLEMENTATION DEFINED. Unimplemented registers are RAZ/WI.

In an implementation that includes StreamID matching, each of these registers is RW. In an implementation that includes StreamID indexing, each register is RAZ/WI.

The format of this register depends on the value of the SMMU_sCR0.EXIDENABLE bit.

When the Extended Stream Matching Extension is implemented, the SMMU_SMRn registers map onto the final 128 SMMU_EXSMRn registers.

When SMMU_sCR2.EXSMRGENABLE is zero, the contents of the SMMU_EXSMR*n* registers, for the appropriate Security state, that are not aliased by SMMU_SMRn, are UNKNOWN/WI.

Attributes 32-bit RW registers with access attributes that depend on the configuration of the implementation. See the configuration details for information.

This register resets to an UNKNOWN value.

See also Table 9-1 on page 9-170.

SMMU_EXSMRn bit assignments

The SMMU_EXSMR*n* bit assignments are identical to those for SMMU_SMRn.

14.2.2 SMMU_EXS2CR*n*, Extended Stream-to-Context Register

The SMMU_EXS2CR*n* characteristics are:

Purpose Specifies an initial context for processing a transaction, where the transaction matches the Stream mapping group that this register belongs to.

Usage constraints An SMMU_EXS2CR*n* register reserved by Secure software must only specify a translation context bank that is reserved for use by Secure software.

An SMMU_EXS2CR*n* register that is accessible from the Non-secure state must only specify a translation context bank that is not reserved by Secure software.

If SMMU_EXS2CR*n*.MTCFG==1, restrictions apply when the memory is either:

- Strongly-ordered or Device, that is, when MemAttr[3:2]==`0b00`.
- Normal Outer Non-cacheable or Inner Non-cacheable, that is, when MemAttr[3:0]==`0b0101`.

In either of these cases, ARM recommends that:

- RACFG is set to `0b11`, indicating No Read-Allocate.
- WACFG is set to `0b11`, indicating No Write-Allocate.
- TRANSIENTCFG is set to `0b10`, indicating Non-transient.
- SHCFG is set to `0b01`, indicating Outer Shareable.

Configurations The format of this register depends on the state of its TYPE[17:16] field. See *SMMU_EXS2CRn bit assignments*.

See also *Multi-format registers and reserved fields on page 9-180*.

When the Extended Stream Matching Extension is implemented, the SMMU_S2CRn registers map onto the final 128 SMMU_EXS2CRn registers.

While SMMU_sCR2.EXSMRGENABLE is zero, the contents of the SMMU_EXS2CR*n* registers, for the appropriate Security state, that are not aliased by SMMU_S2CRn, are UNKNOWN/WI.

The number of SMMU_EXS2CR*n* registers is IMPLEMENTATION DEFINED. Unimplemented registers are RAZ/WI.

Attributes 32-bit RW registers with UNKNOWN reset values. See also Table 9-1 on page 9-170.

SMMU_EXS2CRn bit assignments

The SMMU_EXS2CRn bit assignments are identical for those for SMMU_S2CRn.

Chapter 15
StreamID Compressed Indexing Extension

This chapter describes the StreamID compressed indexing. It contains the following sections:

15.1 About the StreamID Compressed Indexing Extension

The SMMUv2 architecture defines an optional extension to support a different mechanism for StreamID matching, called StreamID Compressed indexing. SMMU_IDR2.COMPINDEXS indicates whether this extension is implemented. A secure read of SMMU_IDR2 returns 1 for this field if the extension is implemented.

The StreamID compressed indexing mechanism is intended to help support systems with a large number of Stream IDs and map these onto the limited set of Stream-to-Context registers, SMMU_S2CRn. It supports StreamIDs of up to 16 bits, see *StreamID size on page 2-52*. That is, it supports a StreamID range of 0-65535 (`0x0000-0xFFFF`).

In an SMMU implementation that supports two Security states, StreamID compressed indexing can be used by only one Security state at a time, and, when the Stream ID Compressed Indexing Extension is implemented:

- SMMU_SCR1.NSCOMPINDEXDISABLE controls which Security state uses StreamID Compressed indexing. This bit resets to zero, corresponding to Non-secure use of StreamID Compressed indexing.
- A Non-secure read of SMMU_IDR2.COMPINDEXS returns 1 if StreamID Compressed indexing is available for use by the Non-secure state.
- The other Security state must use Stream Matching.

Note

Secure state usually only requires a small number of streams, therefore ARM expects that, in a system that uses large Stream IDs:

- Secure state will use Stream matching or Extended Stream matching, if available.
- Non-secure state will use the Compressed indexing mechanism.

For the Security state that has been assigned the StreamID Compressed Indexing extension, as determined by the value of SMMU_SCR1.NSCOMPINDEXDISABLE, the extension is enabled by setting SMMU_sCR2.COMPINDEXENABLE to 1.

15.1.1 StreamID to Stream-to-Context Register indexing

When using StreamID Compressed indexing, the SMMU_COMPINDEX table provides the mapping of a StreamID to the associated Stream-to-Context register, SMMU_S2CRn. This table comprises a set of contiguous 32-bit SMMU_COMPINDEXn registers, each holding four 8-bit S2CRIndex*m* fields, where *m* is in the range 0-3. Each SMMU_COMPINDEXn register supports bytes and register accesses.

The mapping of StreamID *i* to the associated S2CRIndex*m* field, SMMU_COMPINDEXn.S2CRIndex*m*, is defined by:

- $n = i$ DIV 4.
- $m = i$ MOD 4.

Where DIV indicates integer division, and MOD 4 indicates division modulo 4.

SMMU_COMPINDEXn.S2CRIndex*m* holds the number of the associated SMMU_S2CRn.

If the SMMU_COMPINDEXn.S2CRIndex*m* indicates a non-existent SMMU_S2CRn or one that is reserved for use by the opposite Security state, then the transaction is treated as if the transaction did not match, see *No match handling on page 2-57*.

The SMMU_COMPINDEXn registers are UNKNOWN and WI when not enabled. From reset, the values of the SMMU_COMPINDEXn registers are UNKNOWN and therefore the registers must be initialized before they are used to match StreamIDs from incoming transactions.

Note

- Streams remain identified by their associated StreamID. The value of a S2CRIndex field is never used as a stream identifier.
- When using the StreamID Compressed Indexing Extension, the SMMU_SMRn, and, if available, SMMU_EXSMRn are unused by the Security state using the extension.
- The SMMU_COMPINDEXn register table can span multiple pages, but the table is contiguous in address space, independent of PAGESIZE. Thus the address of SMMU_COMPINDEXn is always (8 * PAGESIZE + n * 4).

15.2 StreamID Compressed Indexing Extension registers

The following sections describe the StreamID Compressed Indexing Extension registers:

- *About the StreamID Compressed Indexing Extension registers*.
- *SMMU_COMPINDEXn, Compressed Index register n*.

15.2.1 About the StreamID Compressed Indexing Extension registers

The StreamID Compressed Indexing index is defined as a set of contiguous Compressed Index registers, SMMU_COMPINDEXn. *The global address space on page 8-165* defines the location of these registers.

The SMMU architecture supports 8-bit and 32-bit atomic register accesses to StreamID Compressed Indexing registers. Whether atomic accesses of other sizes are supported is IMPLEMENTATION DEFINED.

15.2.2 SMMU_COMPINDEX*n*, Compressed Index register n

The SMMU_COMPINDEX*n* characteristics are:

Purpose Provides StreamID compressed indexing information.

Usage constraints When the Stream ID Compressed Indexing extension is not enabled, that is the associated SMMU_sCR2.COMPINDEXENABLE is 0, these registers are UNKNOWN and ignore writes.

If the StreamID Compressed Indexing Extension is assigned for use by the Secure state, that is SMMU_SCR1.NSCOMPINDEXDISABLE == 1, then these registers are RAZ/WI to Non-secure accesses.

If SMMU_SCR1.GASRAE is 1, then these registers are RAZ/WI to Non-secure accesses, independent of whether the StreamID Compressed Indexing Extension is assigned for use by the Non-secure state, that is regardless of the value of SMMU_SCR1.NSCOMPINDEXDISABLE.

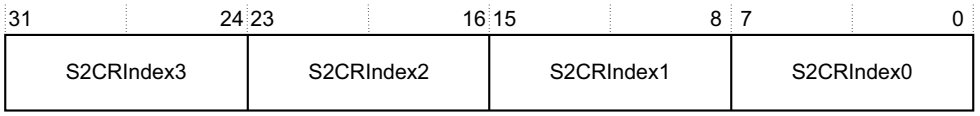
Note

The maximum supported StreamID determines the number of required SMMU_S2CRn registers. If an implementation provides fewer than 64 SMMU_S2CRn registers, then the remainder of the SMMU_S2CRn register space is RES0.

Configurations This register is UNKNOWN after reset.

Attributes A 32-bit RW register that supports byte and 32-bit accesses.

The SMMU_COMPINDEX*n* bit assignments are:



S2CRIndex*m*, for *m* = 0 to 3

StreamID to Context Register Index. Holds the value *n* of the Stream-to-Context register, SMMU_S2CRn, for the StreamID that maps to this S2CRIndex*m* field.

See *StreamID to Stream-to-Context Register indexing on page 15-282* for how the StreamID determines *n* and *m*.

S2CRIndex*m*[7] is RES0.

Note

In an implementation that supports fewer than 64 SMMU_S2CRns, additional high-order bits of S2CRIndex*m* are RES0.

Chapter 16
Stage 1 Translation Context Bank Format

This chapter specifies the format of a stage 1 translation context bank. It contains the following sections:

16.1 Stage 1 translation context bank address space

The stage 1 translation context bank provides registers for initial translation in implementations that support stage 1 translation.

The SMMU architecture supports 32-bit atomic access to all stage 1 translation context bank format registers, and 64-bit atomic access to the 64-bit registers shown in Table 16-1. Whether 8-bit or 16-bit atomic accesses to these registers are supported is IMPLEMENTATION DEFINED.

Table 16-1 shows the address of the stage 1 translation context bank format registers relative to the offset from the translation context bank base address, SMMU_CBn_BASE. Unless otherwise stated, registers are present in any version of the SMMU architecture.

Table 16-1 Stage 1 translation context bank format

Offset	Name	Width	Type	Description
0x00000	SMMU_CBn_SCTLR	32-bit	RW	*SMMU_CBn_SCTLR, System Control Register on page 16-315*
0x00004	SMMU_CBn_ACTLR	32-bit	RW	*SMMU_CBn_ACTLR, Auxiliary Control Register on page 16-294*
0x00008	SMMU_CBn_RESUME	32-bit	WO	Resume or terminate a stalled transaction, *SMMU_CBn_RESUME, Transaction Resume register on page 16-314*
0x0000C	Reserved	-	-	-
0x00010	SMMU_CBn_TCR2	32-bit	RW	*SMMU_CBn_TCR2, Translation Control Register 2 on page 16-336*
0x00014 - 0x0001C	Reserved	-	-	-
0x00020	SMMU_CB*n*_TTBR0	64-bit	RW	*SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339*
0x00028	SMMU_CB*n*_TTBR1	64-bit	RW	*SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339*
0x00030	SMMU_CBn_TCR[a]	32-bit	RW	*SMMU_CBn_TCR, Translation Control Register on page 16-329*
0x00034	SMMU_CBn_CONTEXTIDR	32-bit	RW	*SMMU_CBn_CONTEXTIDR, Context Identification Register on page 16-297*
0x00038	SMMU_CBn_PRRR or SMMU_CBn_MAIR0	32-bit	RW	*SMMU_CBn_PRRR, Primary Region Remap Register on page 16-312* *SMMU_CBn_MAIRm, Memory Attribute Indirection Registers on page 16-304*
0x0003C	SMMU_CBn_NMRR or SMMU_CBn_MAIR1	32-bit	RW	*SMMU_CBn_NMRR, Normal Memory Remap Register on page 16-304* *SMMU_CBn_MAIRm, Memory Attribute Indirection Registers on page 16-304*

Table 16-1 Stage 1 translation context bank format (continued)

Offset	Name	Width	Type	Description
0x00040 - 0x00044	Reserved	-	-	Reserved for IMPLEMENTATION DEFINED memory attributes associated with memory encodings. Such attributes are only additional qualifiers to the memory locations, and cannot change the architected behaviors of the memory attributes defined in the SMMU_CB*n*_MAIR0 and SMMU_CB*n*_MAIR1 registers. See *SMMU_CBn_MAIRm, Memory Attribute Indirection Registers on page 16-304* for more information.
0x00048 - 0x0004C	Reserved	-	-	-
0x00050	SMMU_CBn_PAR	64-bit	RW	*SMMU_CBn_PAR, Physical Address Register on page 16-305*
0x00058	SMMU_CBn_FSR	32-bit	RW	*SMMU_CBn_FSR, Fault Status Register on page 16-298*
0x0005C	SMMU_CBn_FSRRESTORE	32-bit	WO	*SMMU_CBn_FSRRESTORE, Fault Status Restore Register on page 16-301*
0x00060	SMMU_CBn_FAR	64-bit	RW	*SMMU_CBn_FAR, Fault Address Register on page 16-298*
0x00068	SMMU_CB*n*_FSYNR0	32-bit	RW	See *SMMU_CBn_FSYNRm, Fault Syndrome Registers on page 16-301*
0x0006C	SMMU_CB*n*_FSYNR1			
0x00070	Reserved.	-	-	
0x00074	Reserved.	-	-	-
0x00078 - 0x005FC	Reserved	-	-	-
0x00600	SMMU_CBn_TLBIVA	64-bit	WO	*SMMU_CBn_TLBIVA, TLB Invalidate by VA on page 16-322*
0x00608	SMMU_CBn_TLBIVAA	64-bit	WO	*SMMU_CBn_TLBIVAA, TLB Invalidate by VA All ASID on page 16-324*
0x00610	SMMU_CBn_TLBIASID	32-bit	WO	*SMMU_CBn_TLBIASID, TLB Invalidate by ASID on page 16-321*
0x00614	Reserved	-	-	-
0x00618	SMMU_CBn_TLBIALL	32-bit	WO	*SMMU_CBn_TLBIALL, TLB Invalidate All on page 16-321*
0x0061C	Reserved	-	-	-
0x00620	SMMU_CBn_TLBIVAL	64-bit	WO	*SMMU_CBn_TLBIVAL, TLB Invalidate by VA, Last level on page 16-326*
0x00628	SMMU_CBn_TLBIVAAL	64-bit	WO	*SMMU_CBn_TLBIVAAL, TLB Invalidate by VA, All ASID, Last level on page 16-325*
0x00630 - 0x007EC	Reserved	-	-	-
0x007F0	SMMU_CBn_TLBSYNC	32-bit	WO	*SMMU_CBn_TLBSYNC, TLB Synchronize Invalidate on page 16-328*

Table 16-1 Stage 1 translation context bank format (continued)

Offset	Name	Width	Type	Description
0x007F4	SMMU_CBn_TLBSTATUS	32-bit	RO	*SMMU_CBn_TLBSTATUS, TLB Status on page 16-328*
0x007F8 - 0x007FC	Reserved	-	-	-
0x00800	SMMU_CBn_ATS1PR	64-bit	WO	*SMMU_CBn_ATS1PR, Address Translation Stage 1 Privileged Read on page 16-294*
0x00808	SMMU_CBn_ATS1PW	64-bit	WO	*SMMU_CBn_ATS1PW, Address Translation Stage 1 Privileged Write on page 16-295*
0x00810	SMMU_CBn_ATS1UR	64-bit	WO	*SMMU_CBn_ATS1UR, Address Translation Stage 1 Unprivileged Read on page 16-295*
0x00818	SMMU_CBn_ATS1UW	64-bit	WO	*SMMU_CBn_ATS1UW, Address Translation Stage 1 Unprivileged Write on page 16-296*
0x00820 - 0x008EC	Reserved	-	-	-
0x008F0	SMMU_CBn_ATSR	32-bit	RO	*SMMU_CBn_ATSR, Address Translation Status Register on page 16-296*
0x008F4 - 0x00CFC	Reserved	-	-	-
0x00D00 - 0x00DFC	IMPLEMENTATION DEFINED	-	RW	Reserved for IMPLEMENTATION DEFINED purposes
0x00E00 - 0x00E38	SMMU_CBn_PMEVCNTRm	32-bit	RW	*SMMU_CBn_PMEVCNTRm, Performance Monitors Event Counter registers on page 16-311*
0x00E3C - 0x00E7C	Reserved	-	-	-
0x00E80 - 0x00EB8	SMMU_CBn_PMEVTYPERm	32-bit	RW	*SMMU_CBn_PMEVTYPERm, Performance Monitors Event Type Registers on page 16-311*
0x00EBC - 0x00EFC	Reserved	-	-	-
0x00F00	SMMU_CBn_PMCFGR	32-bit	RO	*SMMU_CBn_PMCFGR, Performance Monitors Configuration Register on page 16-310*
0x00F04	SMMU_CBn_PMCR	32-bit	RW	*SMMU_CBn_PMCR, Performance Monitors Control Register on page 16-311*
0x00F08	Reserved	-	-	Reserved for PMUSERENR
0x00F0C - 0x00F1C	Reserved	-	-	-
0x00F20	SMMU_CB*n*_PMCEID0	32-bit	RO	*SMMU_CBn_PMCEIDm, Performance Monitors Common Event Identification registers on page 16-310*
0x00F24	SMMU_CB*n*_PMCEID1			
0x00F28 - 0x00F3C	Reserved	-	-	-
0x00F40	SMMU_CBn_PMCNTENSET	32-bit	RW	*SMMU_CBn_PMCNTENSET, Performance Monitors Count Enable Set register on page 16-310*

Table 16-1 Stage 1 translation context bank format (continued)

Offset	Name	Width	Type	Description
0x00F44	SMMU_CBn_PMCNTENCLR	32-bit	RW	*SMMU_CBn_PMCNTENCLR, Performance Monitors Count Enable Clear register on page 16-310*
0x00F48	SMMU_CBn_PMINTENSET	32-bit	RW	*SMMU_CBn_PMINTENSET, Performance Monitors Interrupt Enable Set register on page 16-312*
0x00F4C	SMMU_CBn_PMINTENCLR	32-bit	RW	*SMMU_CBn_PMINTENCLR, Performance Monitors Interrupt Enable Clear register on page 16-311*
0x00F50	SMMU_CBn_PMOVSCLR	32-bit	RW	*SMMU_CBn_PMOVSCLR, Performance Monitors Overflow Status Clear Register on page 16-312*
0x00F54	Reserved	-	-	Reserved for PMSWINC
0x00F58	SMMU_CBn_PMOVSSET	32-bit	-	*SMMU_CBn_PMOVSSET, Performance Monitors Overflow Status Set Register on page 16-312*
0x00F5C - 0x00FB4	Reserved	-	-	-
0x00FB8	SMMU_CBn_PMAUTHSTATUS	32-bit	RO	*SMMU_CBn_PMAUTHSTATUS, Performance Monitors Authentication Status register on page 16-310*
0x00FBC - 0x00FC8	Reserved	-	-	-
0x00FCC	Reserved	-	-	Reserved for PMDEVTYPE
0x00FD0 - (PAGESIZE −0x4)	Reserved	-	-	-

a. In SMMUv1, this register is SMMU_CBn_TTBCR.

16.2 Reset values

In each SMMU stage 1 translation context bank, the value of each register field is UNKNOWN after a system reset, with the exceptions shown in Table 16-2.

Table 16-2 Predictable post-reset values

Field	Reset value	Notes
SMMU_CBn_ATSR	0	-
SMMU_CBn_FSR.SS	0	No requirement to perform a retry or termination operation.
SMMU_CBn_SCTLR.CFIE	0	Prevent interrupt assertion as a result of UNPREDICTABLE error status.
SMMU_CBn_SCTLR.CFRE	0	Prevent abort as a result of UNPREDICTABLE error status.
SMMU_CBn_TLBSTATUS[a]	0	-

a. In both stage 1 and stage 2 context banks.

16.3 Memory attribute indirection

The SMMU_CBn_MAIRm memory indirection attribute system provides a revised version of the *Type EXtension* (TEX) Remap system to redirect the selection of memory attributes from the translation table entries. This system takes the 3-bit MemAttr field in the translation table entry, and uses it to access an 8-bit field in one of the 32-bit Memory Attribute Indirection Registers, SMMU_CB*n*_MAIR0 and SMMU_CB*n*_MAIR1. See the *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition* and the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile* for more information about the TEX Remap system.

SMMU_CB*n*_MAIR0 is selected when AttrIndex[2] is set to the value 0.

SMMU_CB*n*_MAIR1 is selected when AttrIndex[2] is set to 1.

The field selected in SMMU_CB*n*_MAIR0 and SMMU_CB*n*_MAIR1 is determined by AttrIndex[1:0], as Bits[8×AttrIndx[1:0] + 7:8×AttrIndx[1:0]].

The encoding of each MAIR[7:4] field is shown in Table 16-3, where:

- *R* denotes the Outer Read-Allocate policy.
- *W* denotes the Outer Write-Allocate policy.

The allocation policy is encoded so that a value of 0 means do not allocate, and a value of 1 means allocate.

Table 16-3 MAIR encodings

Bits[7:4]	Meaning
`0b0000`	Strongly-ordered or Device memory.
`0b00`*RW*	Outer Write-Through transient Normal memory. *RW*=`0b00` is not supported.
`0b0100`	Outer Non-cacheable Normal memory.
`0b01`*RW*	Outer Write-Back transient Normal memory. *RW*=`0b00` is not supported.
`0b10`*RW*	Outer Write-Through non-transient Normal memory.
`0b11`*RW*	Outer Write-Back non-transient Normal memory.

The transient variants provide a hint that the data being accessed has a transient access property. The memory system might adjust its handling of the data accordingly.

All other values are UNPREDICTABLE.

Table 16-4 shows the encoding of each MAIR[3:0] field, where:

- *R* denotes the Inner Read-Allocate policy.
- *W* denotes the Inner Write-Allocate policy.

The allocation policy is encoded so that a value of 0 means do not allocate, and a value of 1 means allocate.

Table 16-4 MAIR encodings

Bits[3:0]	Meaning when bits[7:4] == `0b0000`		Meaning when bits[7:4] != `0b0000`
	ARMv8	ARMv7	
`0b0000`	Device-nGnRnE	Strongly-ordered	UNPREDICTABLE.
`0b00`*RW*	UNPREDICTABLE	UNPREDICTABLE	Inner Write-Through transient Normal memory. *RW*=`0b00` is not supported.
`0b0100`	Device-nGnRE	Device	Inner Non-cacheable Normal memory.
`0b01`*RW*	UNPREDICTABLE	UNPREDICTABLE	Inner Write-Back transient Normal memory. *RW*=`0b00` is not supported.

Table 16-4 MAIR encodings (continued)

Bits[3:0]	Meaning when bits[7:4] == 0b0000		Meaning when bits[7:4] != 0b0000
	ARMv8	ARMv7	
0b1000	Device-nGRE	UNPREDICTABLE	Inner Write-Through non-transient Normal memory.
0b10*RW*	UNPREDICTABLE	UNPREDICTABLE	Inner Write-Through non-transient Normal memory.
0b1100	Device-GRE	UNPREDICTABLE	Inner Write-Back non-transient Normal memory.
0b11*RW*	UNPREDICTABLE	UNPREDICTABLE	Inner Write-Back non-transient Normal memory. RW=0b00 is not supported.

All other values are UNPREDICTABLE.

16.4 Multi-format registers and reserved fields

Some registers have a number of possible formats, where the format depends on the format that the register has been configured to, and the format of the data recorded in the register. See *Multi-format registers and reserved fields on page 9-180* for more information, with the exception that in a stage 1 translation context bank, the register format is affected by SMMU_CBn_TCR.EAE instead of by a TYPE field.

16.5 Stage 1 translation context bank register descriptions

This section describes all of the stage 1 translation context bank registers that might be present in an SMMU implementation. Unless otherwise stated, registers are present in any version of the SMMU architecture. Registers are shown in register name order.

16.5.1 SMMU_CB*n*_ACTLR, Auxiliary Control Register

The SMMU_CB*n*_ACTLR characteristics are:

Purpose Provides IMPLEMENTATION SPECIFIC configuration and control options.

Usage constraints No usage constraints apply.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit RW register with an UNKNOWN reset value. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_ACTLR bit assignments are IMPLEMENTATION DEFINED.

16.5.2 SMMU_CB*n*_ATS1PR, Address Translation Stage 1 Privileged Read

The SMMU_CB*n*_ATS1PR characteristics are:

Purpose Translates an argument-supplied input address and writes the result to SMMU_CBn_PAR.

Usage constraints 32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

In SMMUv2, all address translation registers are OPTIONAL. See *Address translation registers in SMMUv2 on page 4-126* for more information.

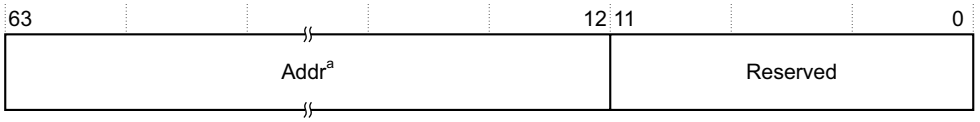
Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

Reads are SBZ.

See also Table 16-1 on page 16-286.

The SMMU_CB*n*_ATS1PR bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

In SMMUv1, Addr is bits[31:12].

Bits[11:0] Reserved.

16.5.3 SMMU_CB*n*_ATS1PW, Address Translation Stage 1 Privileged Write

The SMMU_CB*n*_ATS1PW characteristics are:

Purpose Translates the argument-supplied input address and writes the result to SMMU_CBn_PAR.

Usage constraints 32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

In SMMUv2, all address translation registers are OPTIONAL. See *Address translation registers in SMMUv2 on page 4-126* for more information.

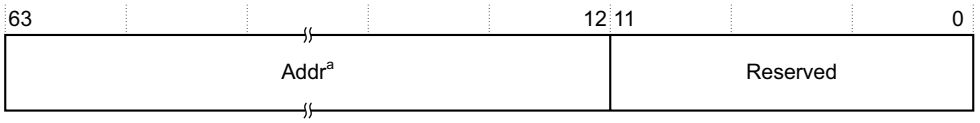
Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

Reads are SBZ.

See also Table 16-1 on page 16-286.

The SMMU_CB*n*_ATS1PW bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

In SMMUv1, Addr is bits[31:12].

Bits[11:0] Reserved.

16.5.4 SMMU_CBn_ATS1UR, Address Translation Stage 1 Unprivileged Read

The SMMU_CB*n*_ATS1UR characteristics are:

Purpose Translates the argument-supplied input address and writes the result to SMMU_CBn_PAR.

Usage constraints 32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

In SMMUv2, all address translation registers are OPTIONAL. See *Address translation registers in SMMUv2 on page 4-126* for more information.

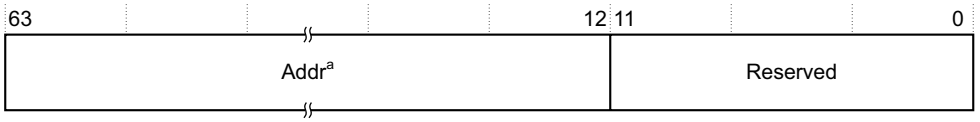
Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

Reads are SBZ.

See also Table 16-1 on page 16-286.

The SMMU_CB*n*_ATS1UR bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

In SMMUv1, Addr is bits[31:12].

Bits[11:0] Reserved.

16.5.5 SMMU_CBn_ATS1UW, Address Translation Stage 1 Unprivileged Write

The SMMU_CB*n*_ATS1UW characteristics are:

Purpose Translates the argument-supplied input address and writes the result to SMMU_CBn_PAR.

Usage constraints 32-bit atomic writes to this register are supported and:

- A 32-bit write to the lower word of the register is zero extended to 64 bits.
- A 32-bit write to the upper word of the register is ignored.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

In SMMUv2, all address translation registers are OPTIONAL. See *Address translation registers in SMMUv2 on page 4-126* for more information.

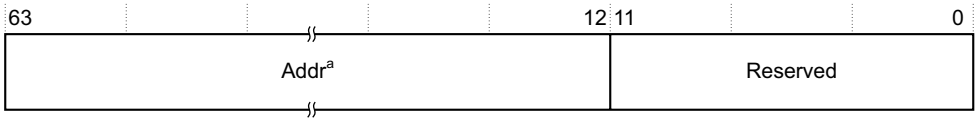
Attributes In SMMUv2, a 64-bit WO register.

In SMMUv1, a 32-bit WO register.

Reads are SBZ.

See also Table 16-1 on page 16-286.

The SMMU_CB*n*_ATS1UW bit assignments are:



a. In SMMUv1, Addr is [31:12]

Addr, bits[63:12]

The input address.

In SMMUv1, Addr is bits[31:12].

Bits[11:0] Reserved.

16.5.6 SMMU_CBn_ATSR, Address Translation Status Register

The SMMU_CB*n*_ATSR characteristics are:

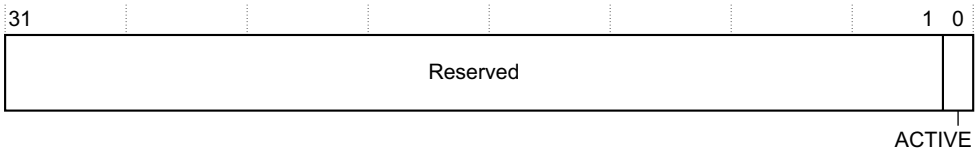
Purpose Provides status information about active address translation operations for a translation context bank.

Usage constraints No usage constraints apply.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit RO register that resets to 0. Writes are ignored. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_ATSR bit assignments are:



Bits[31:1] Reserved.

ACTIVE, bit[0]

Address Translation Active. The possible values of this bit are:

0 Operation not active.

1 Operation active. SMMU_CBn_PAR contents are yet to be updated.

16.5.7 SMMU_CB*n*_CONTEXTIDR, Context Identification Register

The SMMU_CB*n*_CONTEXTIDR characteristics are:

Purpose Identifies the current process identifier and the current address space identifier.

Usage constraints No usage constraints apply.

Configurations When the AArch32 Short-descriptor translation scheme is selected, that is, when the effective value of SMMU_CBn_TCR.EAE is 0, the ASID is used by many memory management functions. The PROCID field has no hardware usage within the SMMU.

When the AArch32 Long-descriptor or the AArch64 translation scheme is selected, that is, when the effective value of SMMU_CBn_TCR.EAE is 1, the ASID field is not present in this register and the PROCID field becomes 32-bit. The MMU refers to the SMMU_CBn_TTBRm.ASID fields instead.

Note

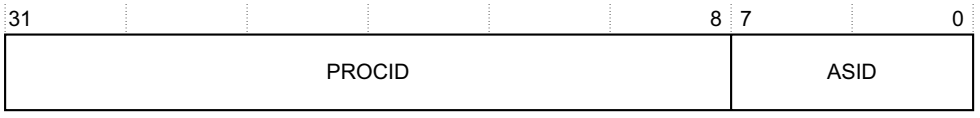
The PROCID field is for OS and debugging purposes only. The SMMU does not use the PROCID field.

It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit RW register with an UNKNOWN reset value. See also Table 16-1 on page 16-286.

Register format when the effective value of SMMU_CBn_TCR.EAE is 0

The SMMU_CB*n*_CONTEXTIDR bit assignments in this format are:



PROCID, bits[31:8]

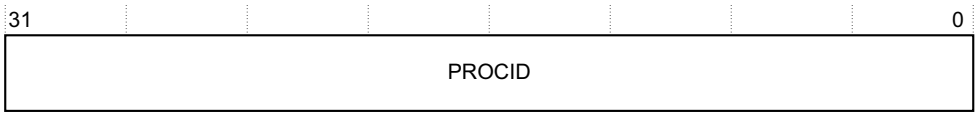
Process Identifier.

ASID, bits[7:0]

Address Space Identifier.

Register format when the effective value of SMMU_CBn_TCR.EAE is 1

The SMMU_CB*n*_CONTEXTIDR bit assignments in this format are:



PROCID, bits[31:0]

Process Identifier.

16.5.8 SMMU_CB*n*_FAR, Fault Address Register

The SMMU_CB*n*_FAR characteristics are:

Purpose Holds the input address of the memory access that caused a synchronous abort exception.

Usage constraints No usage constraints apply.

Configurations For stage 1 and stage 2 translation context banks, the value of N is an IMPLEMENTATION DEFINED value which is not less than the value implied by the value in SMMU_IDR2.UBS.

Unimplemented bits behave as RAZ/WI.

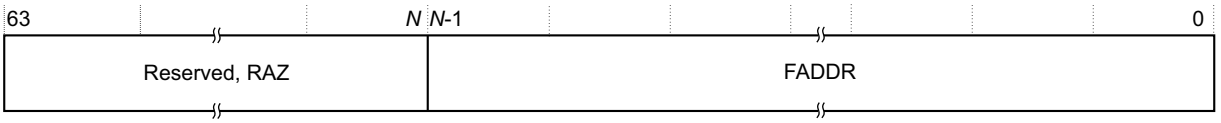
In SMMUv2, the address recorded is always the input address unmodified by sign extension by SMMU_CBn_TCR2.SEP.

Note

The bottom M bits hold no information useful for fixing up the fault. The information is only useful for debug purposes but might have a significant implementation cost. Thus an implementation can make these bits UNKNOWN.

Attributes A 64-bit RW register with an UNKNOWN reset value. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_FAR bit assignments are:



Bits[63:*N*] Reserved.

FADDR, bits[*N*-1:0]

Fault address, the input address of the faulting access.

It is IMPLEMENTATION DEFINED whether the bottom M bits of this field are UNKNOWN, where M=$\log_2$(smallest supported granule size).

```
if SMMU_IDR0.PTFS == 0b0x or SMMU_IDR2.PTFSv8_4KB == 1 then
    M =12;
elseif SMMU_IDR2.PTFSv8_16KB == 1 then
    M = 14;
else
    //There must be at least one granule size supported
    assert (SMMU_IDR2.PTFSv8_64KB == 1);
    M =16;
```

16.5.9 SMMU_CB*n*_FSR, Fault Status Register

The SMMU_CB*n*_FSR characteristics are:

Purpose Provides memory system fault status information.

Usage constraints No usage constraints apply.

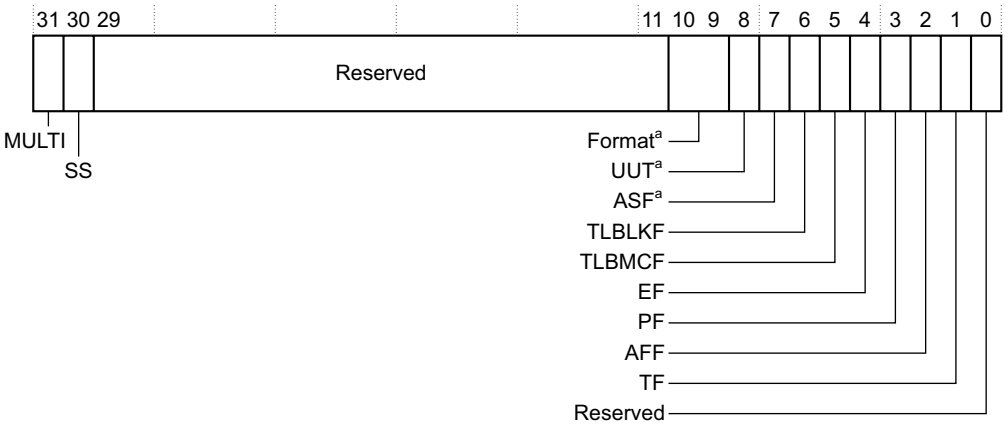
Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit RW register. A value of 1 written to any non-reserved bit clears that bit. A value of 0 written to any of these bits leaves the bit unchanged.

With the exception of the SMMU_CB*n*_FSR.SS bit, the fields in this register have UNKNOWN reset values.

See also Table 16-1 on page 16-286.

The SMMU_CB*n*_FSR bit assignments are:



a. Reserved in SMMUv1

MULTI, bit[31]

Multiple faults, indicates that an additional context fault occurred while the value in SMMU_CB*n*_FSR was nonzero.

SS, bit[30] Stalled Status. The possible values of this read-only bit are:

0 The context is not stalled.

1 The context is stalled because of an exception in the context bank.

When the context is stalled, this bit is set to 1. A write to SMMU_CBn_RESUME is the only way to reset this bit, and depending on the value written, the write either resumes or terminates the stalled transaction.

Note

- This bit is not affected by any SMMU_CBn_FSRRESTORE operation.
- An SMMUv2 implementation might retry a stalled transaction and set this bit to 0 on a write to SMMU_CBARn or SMMU_CBA2Rn, or if a new value is written to SMMU_SCR1.NSNUMCB0.

This bit resets to 0.

Bits[29:11] Reserved.

Format, bits[10:9]

Translation scheme. This field indicates the translation scheme, and therefore the translation table format for which the recorded fault occurred. The possible values for this field are:

`0b00` AArch32 Short-descriptor translation scheme, that is, SMMU_CBA2Rn.VA64 is 0 and the *effective value* of SMMU_CBn_TCR.EAE is 0. This value is not valid for stage 2 translation contexts.

`0b01` AArch32 Long-descriptor translation scheme, that is, SMMU_CBA2Rn.VA64 is 0 and the *effective value* of SMMU_CBn_TCR.EAE is 1.

`0b10` AArch64 translation scheme, that is, SMMU_CBA2Rn.VA64 is 1.

`0b11` Reserved. Treated as AArch64 translation scheme.

This field only has meaning if the context bank is correctly configured as a translating context bank. Otherwise the value of the field is UNKNOWN. For example, if SMMU_CBARn configures a context bank as Stage 1 with Stage 2 Fault then the field is UNKNOWN.

In SMMUv1, this field is reserved.

Note

This field is read-only and is generated from SMMU_CBA2Rn.VA64 and the *effective value* of SMMU_CBn_TCR.EAE. Software must not reconfigure a translation context bank when there are unhandled faults outstanding.

UUT, bit 8 Unsupported Upstream Transaction. The possible values of this bit are:

0 No unsupported upstream transaction fault

1 Unsupported upstream transaction fault recorded.

Note

It is IMPLEMENTATION DEFINED whether the SMMU records a particular unsupported upstream transaction fault.

In SMMUv1, this bit is reserved.

ASF, bit[7] Address size fault. The possible values for this field are:

0 No fault.

1 Fault is a result of an incorrect address size.

In SMMUv1, this bit is reserved.

TLBLKF, bit[6]

TLB lock fault. The possible values of this bit are:

0 There is no TLB lock fault.

1 A TLB lock fault has occurred.

Note

In implementations that do not support TLB locking, this bit might be RAZ/WI.

TLBMCF, bit[5]

TLB match conflict fault. The possible values of this bit are:

0 There is no TLB Match conflict fault.

1 A fault caused by multiple matches was detected in the TLB.

Note

In implementations that do not support TLB match conflict detection, this bit might be RAZ/WI.

EF, bit[4] External fault. The possible values of this bit are:

0 There is no External fault.

1 An External fault has occurred.

Note

In implementations that do not support external fault detection, this bit might be RAZ/WI.

PF, bit[3] Permission fault. The possible values of this bit are:

0 There is no Permission fault.

1 A fault caused by insufficient permission to complete a memory access has occurred.

AFF, bit[2] Access flag fault. The possible values of this bit are:

0 There is no Access flag fault.

1 A fault caused by the access flag being set for the address being accessed has occurred.

TF, bit[1] Translation fault. The possible values of this bit are:

0 There is no Translation fault.

1 A Translation fault has occurred. The mapping for the address being accessed is invalid.

Bit[0] Reserved.

16.5.10 SMMU_CB*n*_FSRRESTORE, Fault Status Restore Register

The SMMU_CB*n*_FSRRESTORE characteristics are:

Purpose Restores the state of SMMU_CBn_FSR, after a reset, for example.

Usage constraints The bit corresponding to the SMMU_CBn_FSR.SS bit, and the bits corresponding to the Format field are ignored. The only way to reset the SS bit is by writing to SMMU_CBn_RESUME.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit WO register. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_FSRRESTORE bit assignments are identical to SMMU_CBn_FSR. With the exception of the Format field and the SS bit, the value of this register overwrites the value of SMMU_CBn_FSR.

16.5.11 SMMU_CBn_FSYNR*m*, Fault Syndrome Registers

The SMMU_CB*n*_FSYNR1 and SMMU_CB*n*_FSYNR0 characteristics are:

Purpose Holds fault syndrome information about the memory access that caused a synchronous abort exception.

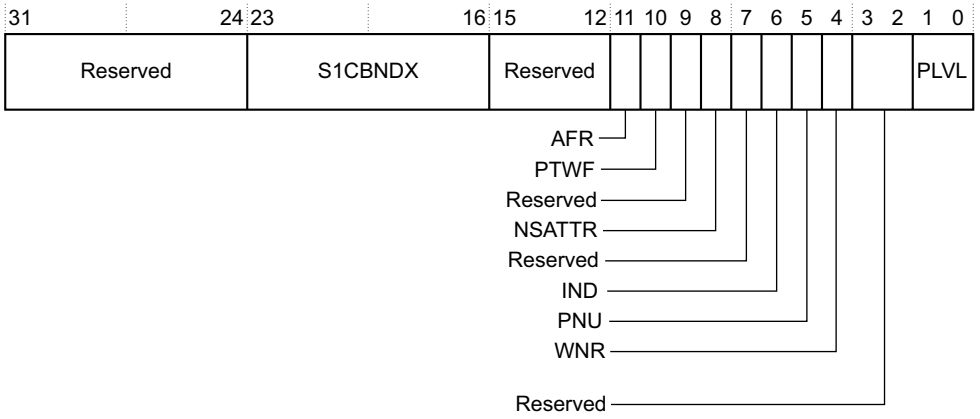
Usage constraints No usage constraints apply.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes 32-bit RW registers with UNKNOWN reset values. See also Table 16-1 on page 16-286.

SMMU_CB*n*_FSYNR0

The SMMU_CB*n*_FSYNR0 bit assignments are:



Bits[31:24] Reserved.

S1CBNDX[23:16]

Stage 1 Context Bank Index associated with the transaction that caused the fault.

For stage 1 followed by stage 2 translation, this field contains the stage 1 translation context bank index for processing the transaction.

For stage 2 only translation, this field is UNKNOWN.

This field is only present in a stage 2 format translation context bank. In a stage 1 format translation context bank, it is UNK/SBPZ.

This field is only valid if SMMU_IDR0.NTS==1. In an implementation that does not include stage 1 followed by stage 2 translation, this field is UNK/WI.

Note

An SMMU implementation is not required to provide storage for all 8 bits of this field, and might provide only enough bits to represent the highest implemented context bank number.

Bits[15:12] Reserved.

AFR, bit[11] Asynchronous Fault Recorded. The possible values of this bit are:

0 A fault was recorded synchronously.

1 A fault was recorded asynchronously.

PTWF, bit[10] Page Table Walk Fault. Indicates whether an external fault was recorded during a translation table access. The possible values of this bit are:

0 . An external fault did not occur while processing a translation table walk.

1 . An external fault occurred while processing a translation table walk.

In SMMUv2, this bit is set to 0 unless an external fault is recorded during a translation table walk.

In SMMUv1, it is UNPREDICTABLE whether this bit is set to 1 for any other fault type.

Bit[9] Reserved.

NSATTR, bit[8]

Non-secure attribute. The possible values of this bit are:

0 The input transaction, after applying SMMU_S2CRn.NSCFG, has a Secure attribute.

1 The input transaction, after applying SMMU_S2CRn.NSCFG, has a Non-secure attribute.

In a Non-secure context bank, this bit is reserved. In a Secure context bank, this bit records the attributes of the transaction after applying the SMMU_S2CRn.NSCFG bit transformation. See *Recording memory attributes on page 3-97* for more information.

Bit[7] Reserved.

IND, bit[6] Instruction Not Data. The possible values of this bit are:

0 Data.

1 Instruction.

This bit records the attributes of the transaction after applying the SMMU_S2CRn.INSTCFG bit transformation. See *Recording memory attributes on page 3-97* for more information.

PNU, bit[5] Privileged Not Unprivileged. The possible values of this bit are:

0 Unprivileged.

1 Privileged.

This bit records the attributes of the transaction after applying the SMMU_S2CRn.PRIVCFG bit transformation. See *Recording memory attributes on page 3-97* for more information.

WNR, bit[4] Write Not Read. The possible values of this bit are:

0 Read.

1 Write.

Note

For far atomic operations, this field indicates the permission that is missing that prevents the far atomic operation from succeeding. If neither permission is available or any other fault occurs, this field will indicate "Read".

Bits[3,2] Reserved.

PLVL, bits[1:0]

Translation Table Level, the level in the translation table walk that the fault is associated with. The encoding of this field is:

`0b00` Level 0. SMMUv2 only.

`0b01` Level 1.

`0b10` Level 2.

`0b11` Level 3.

Faults and translation table level association

The translation table level a fault is associated with is:

- For a fault associated with a translation table walk, the level of table walk being performed.
- For a translation fault, the level of translation table that gave the fault. If a disabled translation table walk causes the fault or if the size of the address presented is out of the range specified for matching with any base address register, the fault is reported as:
  - Level 0, for the AArch64 translation scheme.
  - Level 1, for an AArch32 translation scheme.
- For an access fault, the level of translation table that gave the fault.
- For a permission fault, including a fault caused by a hierarchical permission, the final level of translation table used for that translation.

SMMU_CB*n*_FSYNR1

The 32-bit SMMU_CB*n*_FSYNR1 register bit assignments are IMPLEMENTATION DEFINED.

16.5.12 SMMU_CB*n*_MAIR*m*, Memory Attribute Indirection Registers

The SMMU_CB*n*_MAIR1 and SMMU_CB*n*_MAIR0 characteristics are:

Purpose Provide a revised version of the TEX-Remap system to redirect the selection of memory attributes from the translation table entries.

Usage constraints No usage constraints apply.

Configurations These registers are used when either the AArch32 Long-descriptor or the AArch64 translation scheme is selected. The SMMU_CBn_PRRR and SMMU_CBn_NMRR registers are used when the AArch32 Short-descriptor translation scheme is selected. See *Memory attribute indirection on page 16-291*.

It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes 32-bit RW registers with UNKNOWN reset values. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_MAIR0 bit assignments are:

31 24	23 16	15 8	7 0
Attr3	Attr2	Attr1	Attr0

The SMMU_CB*n*_MAIR1 bit assignments are:

31 24	23 16	15 8	7 0
Attr7	Attr6	Attr5	Attr4

Attr*n* fields For each attribute field, the meaning is as described in *Memory attribute indirection on page 16-291*.

16.5.13 SMMU_CB*n*_NMRR, Normal Memory Remap Register

The SMMU_CB*n*_NMRR characteristics are:

Purpose Provides additional mapping controls for memory regions that are mapped as Normal memory by their entry in SMMU_CBn_PRRR.

Usage constraints No usage constraints apply.

Configurations This register is used when the AArch32 Short-descriptor translation scheme is selected. SMMU_CB*n*_MAIR1 is used when the AArch32 Long-descriptor or the AArch64 translation scheme is selected. See *Memory attribute indirection on page 16-291*.

It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit RW register with an UNKNOWN reset value. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_NMRR bit assignments are:

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
OR7	OR6	OR5	OR4	OR3	OR2	OR1	OR0	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0

OR7 - OR0, bits[31:16]

Outer cacheable property mapping for the memory attributes in *m*, if the region is mapped as Normal memory by the TR*m* entry in SMMU_CBn_PRRR. *m* is the value of the TEX[0], C and B bits in the translation table.

The encoding of this field is:

`0b00` Non-cacheable.

`0b01` Write-Back, Write-Allocate.

`0b10` Write-Through, no Write-Allocate.

`0b11` Write-Back, no Write-Allocate.

IR7 - IR0, bits[15:0]

Inner cacheable property mapping for the memory attributes in *m*, if the region is mapped as Normal memory by the TR*m* entry in SMMU_CBn_PRRR. *m* is the value of the TEX[0], C and B bits in the translation table. The possible values of this field are the same as for those of the SMMU_CB*n*_NMRR.OR*m* fields. The meaning of the field with *m* = 6 is IMPLEMENTATION DEFINED, and might differ from the meaning given in this specification. This is because the meaning of the attribute combination TEX[0] = 1, C = 1, B = 0 is IMPLEMENTATION DEFINED.

16.5.14 SMMU_CB*n*_PAR, Physical Address Register

The SMMU_CB*n*_PAR characteristics are:

Purpose Receives the physical address during any virtual to physical address translation.

Usage constraints No usage constraints apply.

Configurations This register is OPTIONAL and is provided only when the SMMU supports address translation operations, that is, when the SMMU_IDR0.ATOSNS bit is set to 0.

The SMMU_CB*n*_PAR format depends on the translation scheme selected, and whether the transaction completes successfully. See:

- *AArch32 Short-descriptor translation scheme*.
- *AArch32 Long-descriptor and AArch64 translation schemes on page 16-307*.
- *Fault format on page 16-308*.

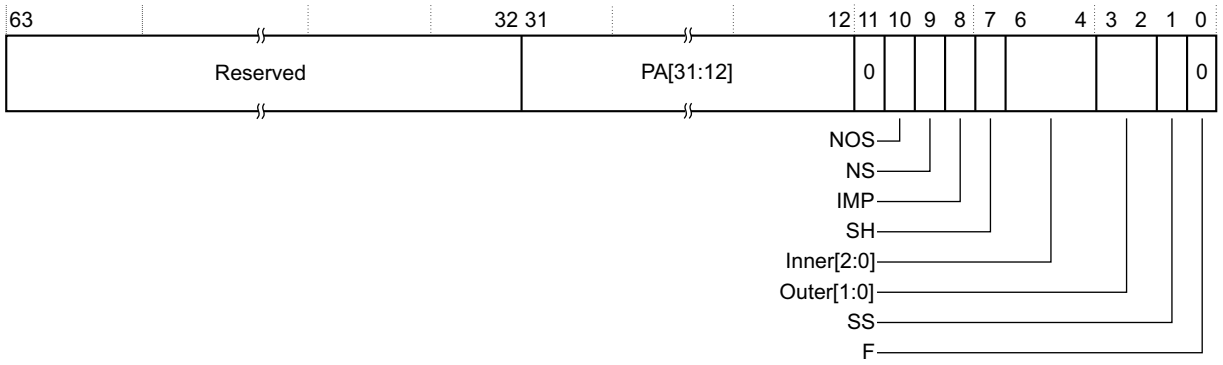
See also *Multi-format registers and reserved fields on page 16-293*.

It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 64-bit RW register with an UNKNOWN reset value. See also Table 16-1 on page 16-286.

AArch32 Short-descriptor translation scheme

When using the AArch32 Short-descriptor translation scheme, if the translation completes successfully, the bit assignments are:



Bits[63:32] Reserved.

PA[31:12], bits[31:12]

Bits[31:12] of the physical address.

Bit[11], 0 Reserved with the value 0.

NOS, bit[10] Not Outer Shareable attribute, indicates whether the physical memory is Outer Shareable. The possible values of this bit are:

0 Memory is Outer Shareable.

1 Memory is not Outer Shareable.

Whether an implementation distinguishes between Inner Shareable and Outer Shareable memory is IMPLEMENTATION DEFINED. If an implementation does not make this distinction, this field is UNK/SBZP.

NS, bit[9] Non-Secure, the NS attribute for a translation table entry read from a Secure translation context bank.

This bit is UNKNOWN for a translation table entry read from a Non-secure translation context bank.

IMP, bit[8] This bit is IMPLEMENTATION DEFINED.

SH, bit[7] Shareable attribute, indicates whether the physical memory is shareable.

The possible values of this bit are:

0 Memory is not shareable

1 Memory is shareable.

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

Inner memory attributes from the translation table entry.

The encoding of this field is:

`0b111` Write-Back, no Write-Allocate.

`0b110` Write-Through.

`0b101` Write-Back, Write-Allocate.

`0b011` Device.

`0b001` Strongly-ordered.

`0b000` Non-cacheable.

All other encodings for Inner[2:0] are reserved.

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

Outer memory attributes from the translation table entry.

The encoding of this field is:

`0b00` Write-Back, no Write-Allocate.

`0b01` Write-Through, no Write-Allocate.

`0b10` Write-Back, Write-Allocate.

`0b11` Non-cacheable.

SS, bit[1] Super Section, indicates if the result is a supersection.

The possible values of this bit are:

0 The page is not a supersection. PAR[31:12] contains PA[31:12], regardless of the page size.

1 The page is part of a supersection, where:

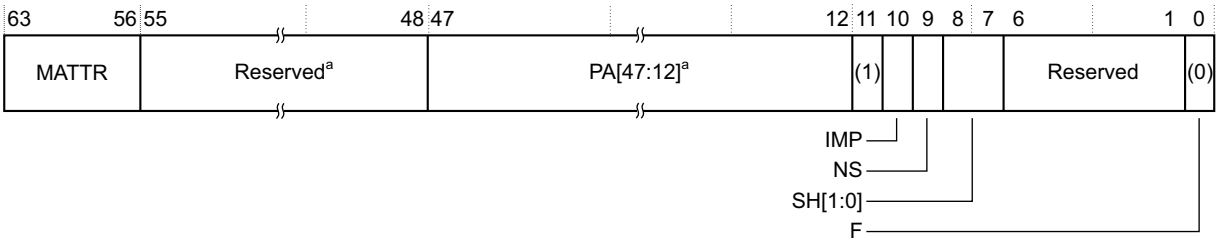
- PAR[31:24] contains PA[31:24].
- PAR[23:16] contains PA[39:32].
- PAR[15:12] contains `0b0000`.

PA[23;12] is the same as VA[23:12] for supersections.

F(0), bit[0] Fault. This bit is 0 if the translation completes successfully.

AArch32 Long-descriptor and AArch64 translation schemes

When using either the AArch32 Long-descriptor or the AArch64 translation scheme, if the translation completes successfully, the bit assignments are:



a. In SMMUv1, bits[47:40] are reserved, and bits[39:12] are PA[39:12]

b. In SMMUv1, bit[11] is reserved.

MATTR, bits[63:56]

Memory Attributes. See *Memory attribute indirection on page 16-291* for more information.

Bits[55:48] Reserved.

PA[47:12], bits[47:12]

Bits[47:12] of the physical address.

When using the AArch64 translation scheme, an illegal physical address might generate an address size fault.

Note

The number of implemented bits is determined by the maximum of SMMU_IDR2.IAS and SMMU_IDR2.OAS.

Unimplemented bits are RAZ/WI.

Bit[11] In SMMUv2, RAO.

In SMMUv1, this bit is reserved.

IMP, bit[10] IMPLEMENTATION DEFINED.

NS, bit[9] Non-Secure, the NS attribute for a translation table entry read from a Secure translation context bank.

This bit is UNKNOWN for a translation table entry read from a Non-secure translation context bank.

For implementations that do not support two Security states, this bit is RAZ/WI.

SH[1:0], bits[8:7]

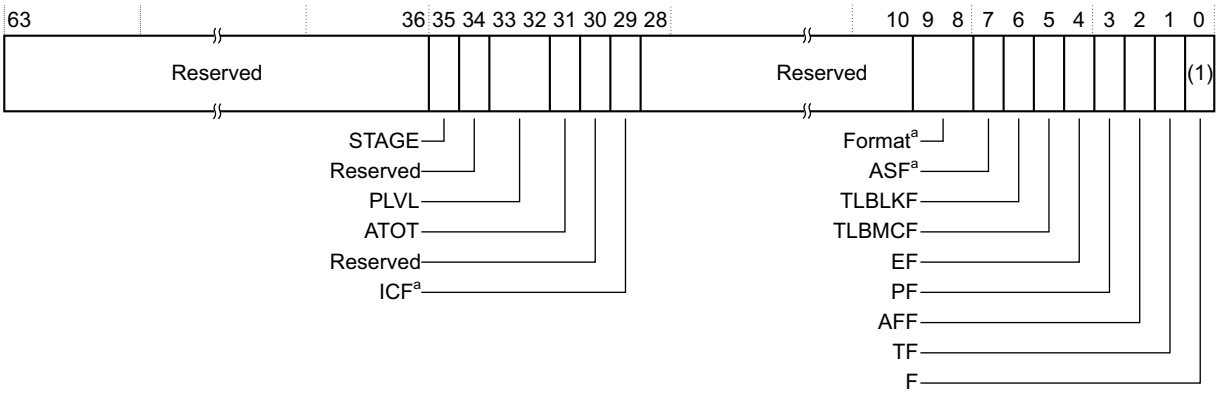
Shareability.

Bits[6:1] Reserved.

F(0) Fault. This bit is 0 if the translation completes successfully.

Fault format

If the translation fails to complete successfully, the bit assignments are, irrespective of translation format:



a. Reserved in SMMUv1

Bits[63:36] Reserved.

STAGE, bit[35]

Stage of translation that encountered the fault. The possible values of this bit are:

0 Stage 1.

1 Stage 2.

Bit[34] Reserved.

PLVL, bits[33:32]

Page level, the level of translation table walk that encountered the fault. The encoding of this field is:

`0b00` Level 0. SMMUv2 only.

`0b01` Level 1.

`0b10` Level 2.

`0b11` Level 3.

ATOT, bit[31] Address Translation Operation Terminated. The possible values of this bit are:

0 This fault does not apply.

1 The address translation operation was terminated. The requested operation could not be completed.

Bit[30] Reserved

ICF, bit[29] Invalid context fault. The possible values of this bit are:

0 This fault does not apply.

1 The transaction mapped to Fault during the address translation operation

In SMMUv1, this bit is reserved.

Bits[28:10] Reserved.

Format, bits[9:8]

Translation scheme. This field indicates the translation scheme, and therefore the translation table format for which the recorded fault occurred. The encoding of this field is:

`0b00` AArch32 Short-descriptor translation scheme, that is, SMMU_CBA2Rn.VA64 is 0 and the *effective value* of SMMU_CBn_TCR.EAE is 0. This value is not valid for stage 2 translation contexts.

`0b01` AArch32 Long-descriptor translation scheme, that is, SMMU_CBA2Rn.VA64 is 0 and the *effective value* of SMMU_CBn_TCR.EAE is 1.

`0b10` AArch64 translation scheme, that is, SMMU_CBA2Rn.VA64 is 1.

`0b11` Reserved. Treated as AArch64 translation scheme.

In SMMUv1, this field is reserved.

Note

This field is read-only and is generated from SMMU_CBA2Rn.VA64 and the *effective value* of SMMU_CBn_TCR.EAE. Software must not reconfigure a translation context bank when there are unhandled faults outstanding.

ASF, bit[7] Address size fault. The possible values for this field are:

0 No fault.

1 Fault is a result of an incorrect address size.

In SMMUv1, this bit is reserved.

TLBLKF, bit[6]

TLB lock fault. The possible values of this bit are:

0 This fault does not apply.

1 TLB lock fault applies.

Note

In implementations that do not support TLB locking, this bit might be RAZ/WI.

TLBMCF, bit[5]

TLB match conflict fault. The possible values of this bit are:

0 This fault does not apply.

1 Fault caused by multiple matches detected in the TLB.

Note

In implementations that do not support TLB match conflict detection, this bit might be RAZ/WI.

EF, bit[4] External fault. The possible values of this bit are:

0 This fault does not apply.

1 Fault caused by an external fault on a translation table walk.

Note

In implementations that do not support external fault detection, this bit might be RAZ/WI.

PF, bit[3] Permission fault. The possible values of this bit are:

0 This fault does not apply.

1 Fault caused by insufficient permission to complete the memory access.

AFF, bit[2] Access flag fault. The possible values of this bit are:

0 This fault does not apply.

1 Access flag fault occurred.

TF, bit[1] Translation fault. The possible values of this bit are:

0 This does not apply.

1 Translation fault condition applies. Invalid mapping for address being accessed.

F(1), bit[0] Fault. This bit is set to 1 if the translation aborts.

16.5.15 SMMU_CB*n*_PMAUTHSTATUS, Performance Monitors Authentication Status register

The Performance Monitors Authentication Status Register, SMMU_CB*n*_PMAUTHSTATUS, provides the equivalent of the PMAUTHSTATUS register, in the register map of a translation context bank.

If a Counter group is revealed in the translation context bank, SMMU_CB*n*_PMAUTHSTATUS has the same format as PMAUTHSTATUS.

If no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMAUTHSTATUS is UNK/SBZP.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.16 SMMU_CB*n*_PMCEID*m*, Performance Monitors Common Event Identification registers

The 32-bit RO Performance Monitors Common Event Identification registers, SMMU_CB*n*_PMCEID0 and SMMU_CB*n*_PMCEID1, provide the equivalent of the SMMU performance monitoring register map PMCEID0 and PMCEID1 registers, in the register map of a translation context bank.

If a Counter group is revealed in a translation context bank, SMMU_CB*n*_PMCEID0 and SMMU_CB*n*_PMCEID1 have the same format as PMCEID0 and PMCEID1.

If no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMCEID0 and SMMU_CB*n*_PMCEID1 are UNK.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.17 SMMU_CB*n*_PMCFGR, Performance Monitors Configuration Register

The 32-bit RO Performance Monitors Configuration Register, SMMU_CB*n*_PMCFGR, provides a performance monitoring configuration register in the register map of a translation context bank.

If a Counter group is revealed in the translation context bank, SMMU_CB*n*_PMCFGR has a similar format to PMCFGR, with the following differences:

- NCG is UNK.
- N indicates the number of event counters in the Counter group.

If no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMCFGR is RAZ/WI. Software can examine SMMU_CB*n*_PMCFGR.SIZE to determine whether any counters are revealed:

- If (SMMU_CB*n*_PMCFGR.SIZE==`0b000000`) no counter group is revealed.
- If (SMMU_CB*n*_PMCFGR.SIZE==`0b011111`) a counter group is revealed.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.18 SMMU_CBn_PMCNTENCLR, Performance Monitors Count Enable Clear register

The Performance Monitors Count Enable Clear register, SMMU_CB*n*_PMCNTENCLR, provides the equivalent of the PMCNTENCLRx register, in the register map of a translation context bank.

If a Counter group is revealed in a translation context bank:

- SMMU_CB*n*_PMCNTENCLR has the same format as PMCNTENCLRx.
- Bit *j* of SMMU_CB*n*_PMCNTENCLR controls event counter *j* in the corresponding Counter group.

If no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMCNTENCLR is UNK/SBZP.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.19 SMMU_CB*n*_PMCNTENSET, Performance Monitors Count Enable Set register

The Performance Monitors Count Enable Set register, SMMU_CB*n*_PMCNTENSET, provides the equivalent of the PMCNTENSETx register, in the register map of a translation context bank.

If a Counter group is revealed in a translation context bank:

- SMMU_CB*n*_PMCNTENSET has the same format as PMCNTENSETx.
- Bit *j* of SMMU_CB*n*_PMCNTENSET controls event counter *j* in the corresponding Counter group.

If a no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMCNTENSET is UNK/SBZP.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.20 SMMU_CB*n*_PMCR, Performance Monitors Control Register

The Performance Monitors Control Register, SMMU_CB*n*_PMCR, provides the equivalent of the PMCR register, in the register map of a translation context bank.

If a Counter group is revealed in the translation context bank, SMMU_CB*n*_PMCR has the same format as PMCR.

If no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMCR is UNK/SBZP.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.21 SMMU_CB*n*_PMEVCNTR*m*, Performance Monitors Event Counter registers

The Performance Monitors Event Counter Registers, SMMU_CB*n*_PMEVCNTR*m*, provide event counter resources in the register map of a translation context bank.

If a Counter group is revealed in a translation context bank:

- The event counter registers of the group are arranged in sequence starting from the first SMMU_CB*n*_PMEVCNTR*m* address offset.
- The SMMU_CB*n*_PMEVCNTR*m* register format is as specified for PMEVCNTRn.
- An SMMU_CB*n*_PMEVCNTR*m* register that corresponds to an offset greater than the number of event counter registers in the Counter group is UNK/SBZP.

If no Counter group is revealed in the translation context bank, all of the SMMU_CB*n*_PMEVCNTR*m* registers are UNK/SBZP.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.22 SMMU_CB*n*_PMEVTYPER*m*, Performance Monitors Event Type Registers

The Performance Monitors Event Type Registers, SMMU_CB*n*_PMEVTYPER*m*, provide event type resources in the register map of a translation context bank.

If a Counter group is revealed in the translation context bank:

- The event type registers of that group are arranged in sequence, starting from the first SMMU_CB*n*_PMEVTYPER*m* address offset.
- The SMMU_CB*n*_PMEVTYPER*m* register format is as specified for PMEVTYPERn.
- An SMMU_CB*n*_PMEVTYPER*m* register that corresponds to an offset greater than the number of event type registers in the Counter Group is UNK/SBZP.

If no Counter group is revealed in the translation context bank, all of the SMMU_CB*n*_PMEVTYPER*m* registers are WI.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.23 SMMU_CBn_PMINTENCLR, Performance Monitors Interrupt Enable Clear register

The Performance Monitors Interrupt Enable Clear Register, SMMU_CB*n*_PMINTENCLR, provides the equivalent of the PMINTENCLRx register, in the register map of a translation context bank.

If a Counter group is revealed in the translation context bank:

- SMMU_CB*n*_PMINTENCLR has the same format as PMINTENCLRx.
- Bit *j* of SMMU_CB*n*_PMINTENCLR corresponds to event counter *j* in the corresponding Counter group.

If no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMINTENCLR is UNK/SBZP.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.24 SMMU_CB*n*_PMINTENSET, Performance Monitors Interrupt Enable Set register

The Performance Monitors Interrupt Enable Set register, SMMU_CB*n*_PMINTENSET, provides the equivalent of the PMINTENSETx register, in the register map of a translation context bank.

If a Counter group is revealed in the translation context bank:

- SMMU_CB*n*_PMINTENSET has the same format as PMINTENSETx.
- Bit *j* of SMMU_CB*n*_PMINTENSET corresponds to event counter *j* in the corresponding Counter group.

If no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMINTENSET is UNK/SBZP.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.25 SMMU_CB*n*_PMOVSCLR, Performance Monitors Overflow Status Clear Register

The Performance Monitors Overflow Status Clear Register, SMMU_CB*n*_PMOVSCLR, provides the equivalent of the PMOVSCLRx register, in the register map of a translation context bank.

If a Counter group is revealed in the translation context bank:

- SMMU_CB*n*_PMOVSCLR has the same format as PMOVSCLRx.
- Bit *j* of SMMU_CB*n*_PMOVSCLR corresponds to event counter *j* in the corresponding Counter group.

If no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMOVSCLR is UNK/SBZP.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.26 SMMU_CB*n*_PMOVSSET, Performance Monitors Overflow Status Set Register

The Performance Monitors Overflow Status Set Register, SMMU_CB*n*_PMOVSSET, provides the equivalent of PMOVSSETx, in the register map of a translation context bank.

If a Counter group is revealed in the translation context bank:

- SMMU_CB*n*_PMOVSSET has the same format as PMOVSSETx.
- Bit *j* of SMMU_CB*n*_PMOVSSET corresponds to event counter *j* in the corresponding Counter group.

If no Counter group is revealed in the translation context bank, SMMU_CB*n*_PMOVSSET is UNK/SBZP.

For more information, see Chapter 6 *SMMU Performance Monitors Extension*.

16.5.27 SMMU_CB*n*_PRRR, Primary Region Remap Register

The SMMU_CB*n*_PRRR characteristics are:

Purpose Controls top-level mapping of the TEX[0], C, and B memory region attributes.

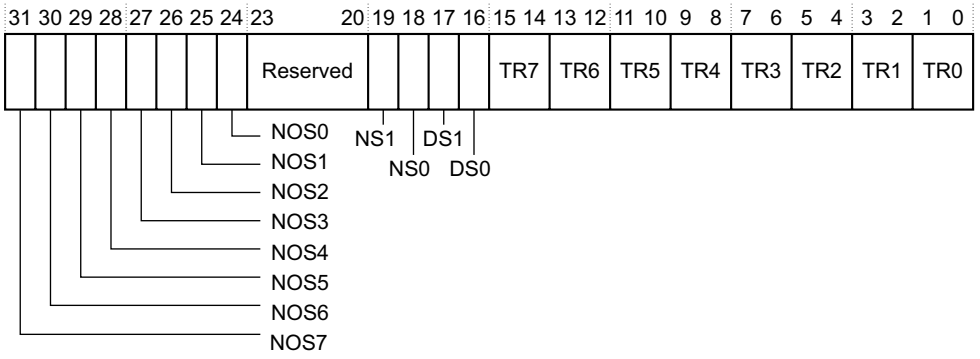
Usage constraints No usage constraints apply.

Configurations This register is used when the AArch32 Short-descriptor translation scheme is selected. SMMU_CB*n*_PRRR is used when the AArch32 Long-descriptor or the AArch64 translation scheme is selected. See *Memory attribute indirection on page 16-291*.

It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit RW register with an UNKNOWN reset value. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_PRRR bit assignments are:



NOS*m*, bits[31:24]

Outer Shareable property mapping for memory attributes in *n*, if the region is mapped as Normal memory that is Shareable where *n* is the value of the TEX[0] C and B bits in the translation table.

The possible value of each NOS bit is:

0 If the region is mapped as shareable Normal memory, the region is Outer Shareable.

1 The region is not Outer Shareable.

The meaning of the field where *m* = 6 is IMPLEMENTATION DEFINED and might differ from the meaning given in this specification. This is because the meaning of the attribute combination TEX[0] = 1, C = 1, B = 0 is IMPLEMENTATION DEFINED. If the implementation does not distinguish between Inner Shareable and Outer Shareable, these bits are reserved and are RAZ/WI.

Bits[23:20] Reserved.

NS1, bit[19] Mapping of the S = 1 attribute for Normal memory. This bit gives the mapped Shareable attribute for a region of memory that is mapped as Normal memory and has the S bit set to the value 1.

The possible values of this bit are:

0 The region is not shareable.

1 The region is shareable.

NS0, bit[18] Mapping of S = 0 attribute for Normal memory. This bit gives the mapped Shareable attribute for a region of memory that is mapped as Normal memory and has the S bit set to the value 0.

The possible values of this bit are:

0 The region is not shareable.

1 The region is shareable.

DS1, bit[17] Mapping of S = 1 attribute for Device memory. This bit gives the mapped Shareable attribute for a region of memory that is mapped as Device memory and has the S bit set to the value 0.

The possible values of this bit are:

0 The region is not shareable.

1 The region is shareable.

DS0, bit[16] Mapping of S = 0 attribute for Device memory. This bit gives the mapped Shareable attribute for a region of memory that is mapped as Device memory and has the S bit set to the value 0.

The possible values of this bit are:

0 The region is not shareable.

1 The region is shareable.

TR*k*, bits[15:0] Primary TEX mapping for memory attributes *m*, where *m* is the value of the TEX[0] C and B bits in the translation table entry. This field defines the mapped memory type for a region with attributes *n*.

The encoding of this field is:

`0b00` Strongly-ordered.

`0b01` Device.

`0b10` Normal memory.

`0b11` Reserved. Effect is UNPREDICTABLE.

The meaning of the field with $n = 6$ is IMPLEMENTATION DEFINED and might differ from the meaning given here. This is because the meaning of the attribute combination TEX[0] = 1, C = 1, B = 0 is IMPLEMENTATION DEFINED.

Table 16-5 shows the mapping between the memory region attributes and the value *n* used in the SMMU_CB*n*_PRRR.NOS*n* and SMMU_CB*n*_PRRR.TR*n* field descriptions.

Table 16-5 TEX[0] mappings

TEX[0]	C	B	*n* value
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

16.5.28 SMMU_CB*n*_RESUME, Transaction Resume register

The SMMU_CB*n*_RESUME characteristics are:

Purpose Resumes or terminates the operation of a stalled transaction.

The correct software sequence for resuming a transaction after a reported fault is as follows:

1. Software corrects the conditions that caused the fault as reported in SMMU_CBn_FSR.
2. Software writes to SMMU_CBn_FSR to clear the appropriate fault bits.
3. Software writes to SMMU_CBn_RESUME.

Failure to follow this sequence might result in a new fault and SMMU_CBn_FSR.MULTI becoming set to 1.

This is the primary mechanism for clearing the SMMU_CBn_FSR.SS bit. An SMMUv2 implementation might also retry transactions and clear the SS bit on writes to SMMU_CBARn or SMMU_CBA2Rn. Additionally, an SMMUv2 implementation might retry a transaction in all context banks affected by a change to SMMU_SCR1.NSNUMCBO.

If a stalled transaction is retried by writing 0 to this register, the retry always occurs before the first use of SMMU_CBn_TCR. For stage 1 followed by stage 2 translations stalled in stage 2, the translation restarts in stage 1.

This means that a retried transaction is guaranteed to be affected by changes to the SMMU_CBn_TCR. However it is not guaranteed to be affected by changes to registers such as SMMU_S2CRn, or SMMU_CBARn.

A resumed transaction retains its original SSD security state, and cannot be affected by changes to SMMU_SSDRn.

Note

Optionally, an implementation can retry a transaction from an earlier stage of the translation process. This means that in SMMUv1, transactions can be subject to SSD again. However, in SMMUv2, the security state of a transaction must be preserved throughout the lifetime of the transaction, and the earliest point from which a transaction can be retried is immediately after SSD.

If the value of SMMU_CBn_SCTLR.CFCFG changes the current configuration to the Terminate fault model while a translation context bank is stalled, the SMMU_CBn_FSR.SS bit is unaffected and software must write to SMMU_CBn_RESUME to terminate or resume the stalled transaction.

If a transaction resumes, the latest point in the translation process that it resumes from is the first use of SMMU_CBn_TCR. For stage 1 followed by stage 2 translations stalling in stage 2, the translation restarts in stage 1. An implementation can choose to restart at an earlier point. For example, an implementation might choose to restart after the SSD step.

See *Handling a context fault on page 3-106* for more information about writing to this register.

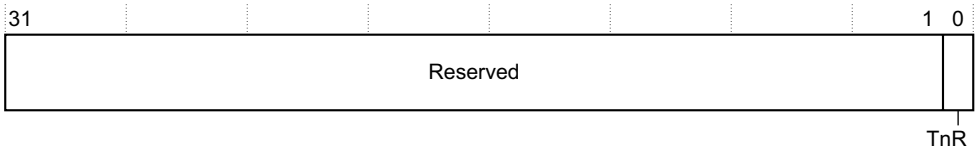
Usage constraints See *Context fault model pseudocode on page 3-108* for information about behavior when the SMMU_CBn_SCTLR.CFCFG bit is set to 0, indicating the Terminate model, while there is a stalled transaction.

If the implementation does not support the stall model, or if the translation context bank is not stalled, writes to this register are ignored.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit WO register. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_RESUME bit assignments are:



Bits[31:1] Reserved.

TnR, bit[0] Terminate not Retry.

The possible values of this bit are:

0 Retry the stalled transaction.

1 Terminate the stalled transaction.

16.5.29 SMMU_CB*n*_SCTLR, System Control Register

The SMMU_CB*n*_SCTLR characteristics are:

Purpose Provides top-level control of the translation system for the related translation context bank.

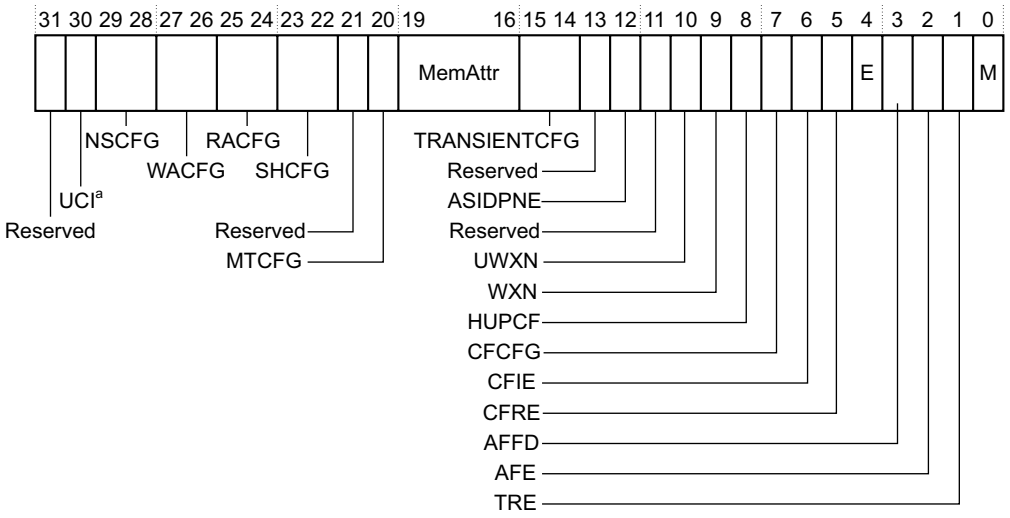
Usage constraints An SMMU implementation might configure certain attributes as RAZ/WI if the downstream bus system does not support this bus marking. See *Memory type and shareability attribute determination on page 2-59* for more information.

Configurations See the field descriptions for field-specific configuration details.

It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit RW register. With the exception of SMMU_CB*n*_SCTLR[6:5], the fields in this register have UNKNOWN reset values. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_SCTLR bit assignments are:



a. In SMMUv1, reserved. In SMMUv2, applies only when SMMU_CBA2Rn.VA64==1, otherwise ignored.

Bits [31] Reserved.

UCI, bit[30] User Cache Maintenance Operation Enable.

The possible values of this bit are:

0 Cache maintenance operations from EL0 are disabled.

1 Cache maintenance operations from EL0 are enabled.

In SMMUv2, this bit applies only when the context bank is using the AArch64 translation scheme. This bit is ignored when the SMMU_CBA2Rn.VA64 bit is set to 0.

In SMMUv1, this bit is reserved.

NSCFG, bits[29:28]

Non-Secure Configuration, controls the Non-secure attribute for any transaction where the translation context bank translation is disabled. That is, where SMMU_CB*n*_SCTLR.M==0.

SMMU_CB*n*_SCTLR.NSCFG exists only in translation context banks reserved by Secure software. In Non-secure translation context banks, this field is UNK/SBZP.

The encoding of this field is:

`0b00` Default Non-secure attribute.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Secure.

`0b11` Non-secure.

WACFG, bits[27:26]

Write-Allocate Configuration, controls the allocation hint for a write transaction where the translation context bank translation is disabled. That is, where SMMU_CB*n*_SCTLR.M==0.

The encoding of this field is:

`0b00` Default allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Write-Allocate.

`0b11` No Write-Allocate.

In an implementation that does not support Write-Allocate configuration, this field is RAZ/WI.

RACFG, bits[25:24]

Read-Allocate Configuration, controls the allocation hint for a read transaction where the translation context bank translation is disabled. That is, where SMMU_CB*n*_SCTLR.M==0.

The encoding of this field is:

`0b00` Default allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Read-Allocate.

`0b11` No Read-Allocate.

In an implementation that does not support Read-Allocate configuration, this field is RAZ/WI.

SHCFG, bits[23:22]

Shared Configuration, controls the shareable attribute of a transaction where the translation context bank is disabled. That is, where SMMU_CB*n*_SCTLR.M==0.

The encoding of this field is:

`0b00` Default shareable attribute.

`0b01` Outer Shareable.

`0b10` Inner Shareable.

`0b11` Non-shareable.

In an implementation that does not support Shared configuration, this field is RAZ/WI.

Bit[21] Reserved.

MTCFG, bit[20]

Memory Type Configuration, controls the memory type attribute of a transaction where the translation context bank translation is disabled. That is, where SMMU_CB*n*_SCTLR.M==0.

The possible values of this bit are:

0 Default memory attributes.

1 Use the MemAttr field for memory attributes.

MemAttr, bits[19:16]

Memory Attributes. These attributes are permitted to be overlaid if SMMU_CB*n*_SCTLR.M==0 and SMMU_CB*n*_SCTLR.MTCFG==1.

Memory attribute, MemAttr on page 9-179 describes the MemAttr field encoding.

TRANSIENTCFG, bits[15:14]

Transient Allocate Configuration, controls the transient allocation hint.

The encoding of this field is:

`0x00` Default transient allocation attributes.

`0x01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0x10` Non-transient.

`0x11` Transient.

This field applies where SMMU_CBn_SCTLR.M== 0.

It is IMPLEMENTATION DEFINED whether this field is present. If not implemented, for examples, in bus systems that do not support transient marking, these bits behave as RAZ/WI.

Bit[13] Reserved.

ASIDPNE, bit[12]

Address Space Identifier Private Namespace Enable.

The possible values of this bit are:

0 The SMMU ASID values for this translation context bank are coordinated with the wider system.

1 The SMMU ASID values for this translation context bank are a private namespace that is not coordinated with the wider system.

This bit is a hint. When set to 1 it indicates that the SMMU can maintain a private ASID space for each VMID.

Note

- The SMMU specification does not require SMMU implementations to support these private ASID spaces, The specification permits an implementation to always behave as if the value of the ASIDPNE bit is 0 for all purposes other than reading or writing the value of the bit.
- On an SMMU implementation that supports private ASID spaces, the value of the ASIDPNE bit affects TLB operation. How the TLBs support private ASID spaces is IMPLEMENTATION DEFINED. The SMMU architecture specifies required and optional TLB invalidation effects, but all other aspects of TLB implementation are outside the scope of the SMMU architecture.

When supported, these private ASID spaces are not subject to broadcast TLB invalidation operations. A TLB entry that is allocated when the value of ASIDPNE is 1 can contain a hint that the entry can be excluded from broadcast TLB invalidation operations that target that ASID value.

When the value of ASIDPNE is 1, a broadcast TLB Invalidate operation specifying an ASID value is not required to apply to any cached translations in the SMMU for which the value of ASIDPNE was 1 at the time of allocation. A Broadcast TLB Invalidate operation is still permitted to affect such cached translation in the SMMU and is permitted to apply to all unlocked entries.

The scope of the ASID namespace for a translation context bank is limited to the VMID associated with that translation context bank in the Stream-to-context mapping process.

Bit[11] Reserved.

UWXN, bit[10]

Unprivileged Writable Execute Never.

The possible values of this bit are:

0 . This behavior is not enabled.

1 . Raise a stage 1 permission fault if an instruction fetch occurs from a memory location that permits writes for unprivileged accesses.

If SMMU_IDR0.PTFS indicates no AArch32 support then an implementation may choose to make this field RAZ/WI.

Note

This field only applies to translated transactions. That is, when SMMU_CB*n*_SCTLR.M==1.

WXN, bit[9] Writable Execute Never.

The possible values of this bit are:

0 . This behavior is not enabled.

1 . Raise a stage 1 permission fault if an instruction fetch occurs from a memory location that permits writes.

Note

This field only applies to translated transactions. That is, when SMMU_CB*n*_SCTLR.M==1.

HUPCF, bit[8] Hit under previous context fault.

The possible values of this bit are:

0 Stall or terminate subsequent transactions in the presence of an outstanding context fault.

1 Process all subsequent transactions independently of any outstanding context fault.

Note

In SMMUv2, if this bit is set to 0, it is not guaranteed that faults from different devices are recorded in absolute temporal order. When a device faults, although the setting of this bit is observed for that device, the ordering of suspending transactions from different non-faulting devices is not guaranteed.

See *Behavior when stage 2 transactions terminate on a fault on page 3-98* for more information about this bit.

CFCFG, bit[7] Context fault configuration.

The possible values of this bit are:

0 Terminate.

1 Stall.

CFIE, bit[6] Context fault interrupt enable.

The possible values of this bit are:

0 Do not raise an interrupt when a context fault occurs.

1 Raise an interrupt when a context fault occurs.

This bit resets to 0.

CFRE, bit[5] Context fault report enable.

The possible values of this bit are:

0 Do not return an abort when a context fault occurs.

1 Return an abort when a context fault occurs.

This bit resets to 0.

For exclusive access transactions, when this bit is set to 0, ARM recommends that the SMMU reports the transaction as failed. See *Reporting exclusive access transactions on page 3-122* for more information.

E, bit[4] Endianness, indicates the endian format of translation table entries.

The possible values of this bit are:

0 Little endian format.

1 Big endian format.

AFFD, bit[3] Access flag fault disable, determines whether Access flag faults are enabled. This bit only applies when AFE==1, or when the effective value of AFE is 1, that is, if the effective SMMU_CBn_TCR.EAE==1.

The possible values of this bit are:

0 Access flag faults are enabled.

1 Access flag faults are disabled.

If AFFD==0, AP[0]==0 in the translation table entry causes an Access flag fault, which is reported by SMMU_CBn_FSR.

If AFFD==1, hardware behaves as if AP[0]==1, regardless of the translation table entry value.

If the SMMU shares translation tables with the processor, any descriptor configured with AP[0] as an access flag, that is, when the effective value of AFE is 1, is not held in a processor TLB entry. However, such descriptors might be held in SMMU TLBs. Therefore, any software that modifies descriptors must be aware that an obsolete descriptor might be cached in the SMMU TLB.

For more information about the *Access permission* (AP) bit, see the *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

It is IMPLEMENTATION DEFINED whether this bit is cacheable in the TLB

Note

ARM recommends that software issues TLB maintenance operations if this bit is modified.

AFE, bit[2] Access flag enable, enables the AP[0] bit in the translation table descriptors to be used as an access flag.

The possible values of this bit are:

0 In the translation table descriptors, AP[0] is an access permissions bit. The full range of access permissions is supported. No access flag is implemented.

1 In the translation table descriptors, AP[0] is an access flag. Only the simplified model for access permissions is supported.

If the AArch32 Long-descriptor or the AArch64 translation scheme is selected, that is, if SMMU_CBn_TCR.EAE is set to 1, this bit has no effect and the SMMU behaves as if the bit is set. ARM recommends that software treats this bit as UNK/SBOP when SMMU_CBn_TCR.EAE==1.

TRE, bit[1] TEX Remap Enable, enables remapping of the TEX[2:1] bits for use as two translation table bits that can be managed by the operating system. Enabling this remapping also changes the scheme that describes the memory region attributes in the VMSA.

The possible values of this bit are:

0 TEX Remap is disabled. Bits TEX[2:0] are used with the C and B bits to describe the memory region attributes.

1 TEX Remap is enabled. Bits TEX[2:1] are reassigned for use as flags managed by the operating system. The TEX[0], C and B bits describe the memory region attributes, with the MMU remap registers.

If the AArch32 Long-descriptor or AArch64 translation scheme is selected, that is, if SMMU_CBn_TCR.EAE is set to 1, this bit has no effect and the SMMU behaves as if the bit is set. ARM recommends that software treats this bit as UNK/SBOP when SMMU_CBn_TCR.EAE has the value 1.

M, bit[0] MMU Enable, a global enable bit for the involved translation context bank.

The possible values of this bit are:

0 MMU behavior for this translation context bank is disabled.

1 MMU behavior for this translation context bank is enabled.

This field is not cacheable in TLB entries.

When this bit is changed:

- Any transaction that arrives at the SMMU after the write that updated SMMU_CBn_SCTLR.M completed is guaranteed to use the new value.
- Any transaction that arrived at the SMMU before the completion of the update of SMMU_CBn_SCTLR.M might be processed using either the old value or the new value.

Note

- A TLB invalidate sequence that invalidates all transactions from this bank that might be in progress, and is issued after a completed SMMU_CBn_SCTLR.M write that changes the value from 1 to 0, will complete only after the completion of all non-stalled transactions that were processed using M==1.
- There is no mechanism by which software can check for the completion of stalled transactions that were part-processed using M==1 before they stalled.

16.5.30 SMMU_CB*n*_TLBIALL, TLB Invalidate All

The SMMU_CB*n*_TLBIALL characteristics are:

Purpose Invalidates all unlocked TLB entries that are tagged as:

- Hypervisor, for HYPC banks.
- Non-secure, using the VMID of the context bank, for Non-secure, non-HYPC context banks.
- Secure, using any ASID, for Secure context banks.

Usage constraints This operation requires no arguments.

For Non-secure, non-HYPC context banks, this operation only has to apply to TLB entries associated with the VMID used for stage 1 translation context bank.

If SMMU_CBARn.HYPC has the value 1, this operation only has to apply to TLB entries associated with hypervisor contexts. The VMID is therefore irrelevant to the operation. See *Hypervisor contexts (HYPC) on page 2-86* for more information.

If the SMMU_CBA2Rn.MONC bit is set to 1, this operation only has to apply to TLB entries associated with Monitor contexts.

In an implementation that supports two Security states, this operation only has to apply to TLB entries associated with the Security state of the stage 1 translation context bank. The VMID of Secure context banks is ignored, even in implementations that support a VMID for Secure banks. See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

See *TLB maintenance operations on page 5-137* for more details about the usage model.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit WO register. Reads are SBZ. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_TLBIALL register bit assignments are reserved.

16.5.31 SMMU_CB*n*_TLBIASID, TLB Invalidate by ASID

The SMMU_CB*n*_TLBIASID characteristics are:

Purpose Invalidates all unlocked TLB entries that match the ASID provided as an argument.

Usage constraints For Non-secure, non-HYPC context banks, this operation only has to apply to unlocked non-global TLB entries that match the VMID used for the stage 1 translation context bank.

If SMMU_CBARn.HYPC is 1, this operation is UNPREDICTABLE.

If SMMU_CBA2Rn.MONC is 1, this operation is UNPREDICTABLE.

In an implementation that supports two Security states, this operation only has to apply to unlocked TLB entries associated with the Security state of the stage 1 translation context bank.

See *TLB maintenance operations on page 5-137* for more details about the usage model.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit WO register. Reads are SBZ. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_TLBIASID bit assignments are:

Bits[31:N] Reserved.

In SMMUv2, N = 16.

In SMMUv1, N = 8.

ASID, bits[N–1:0]

Address Space Identifier, the input to the invalidation operation.

In SMMUv2:

- For an AArch32 translation scheme, bits[15:8] are treated as 0.
- For the AArch64 translation scheme, if SMMU_CBn_TCR2.AS==0, ASID bits[15:8] must be written as 0.

In SMMUv1, ASID is [7:0].

16.5.32 SMMU_CB*n*_TLBIVA, TLB Invalidate by VA

The SMMU_CB*n*_TLBIVA characteristics are:

Purpose Invalidates all unlocked TLB entries that match both the VA provided.

Usage constraints For Non-secure, non-HYPC context banks, this operation only has to apply to unlocked TLB entries associated with the ASID and VMID used for the stage 1 translation context bank. The ASID is not checked for global entries in the TLB.

The VMID of Secure context banks is ignored for TLB matching purposes, even in implementations that support a VMID for Secure banks.

See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

If SMMU_CBARn.HYPC has the value 1, this operation only has to apply to unlocked TLB entries associated with hypervisor contexts. The VMID and ASID are therefore irrelevant to the operation. See *Hypervisor contexts (HYPC) on page 2-86* for more information.

In an implementation that supports two Security states, this operation is only required to apply to unlocked TLB entries associated with the Security state of the stage 1 translation context bank.

There are two register formats for this register, AArch32 or AArch64, dependent on SMMU_CBA2Rn.VA64:

- If SMMU_CBA2Rn.VA64 is one, then AArch64 format is selected. The programmer should use 64 bit accesses to this register. If 32-bit accesses are used then writes to the top 32 bits are ignored and writes to the lower 32 bits are zero extended.
- If SMMU_CBA2Rn.VA64 is zero, then AArch32 format is selected. The programmer should use 32 bit accesses to this register. If 64-bit accesses are used then the top 32 bits are ignored.

See *TLB maintenance operations on page 5-137* for more details about the usage model.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

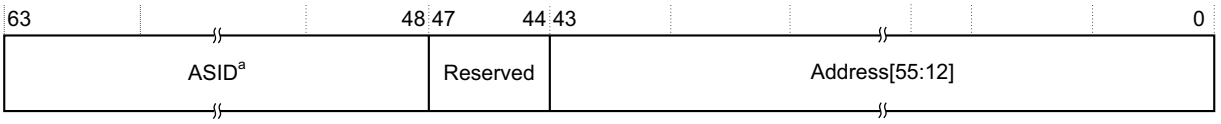
Attributes In AArch64, a 64-bit WO register.

In AArch32, a 32-bit WO register.

Reads are SBZ. See also Table 16-1 on page 16-286.

SMMU_CBn_TLBIVA bit assignments, AArch64 translation scheme

The SMMU_CB*n*_TLBIVA bit assignments for AArch64 are:



a. For operations that do not require an ASID, this field is reserved.

ASID, bits[63:48]

Address Space Identifier, the input to the invalidation operation.

When SMMU_CBn_TCR2.AS==0, the upper 8 bits of this field must be written as 0.

This field is reserved in HYPC and MONC banks and must be written as zero.

Bits[47:44] Reserved.

Address[55:12], bits[43:0]

Bits[55:12] of the address to be invalidated:

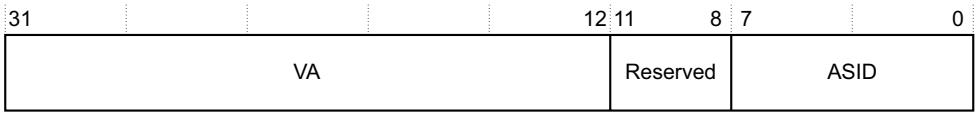
- If the translation granule is 64KB, then bits corresponding to Address[15:12] are ignored.
- If the translation granule is 16KB, then bits corresponding to Address[13:12] are ignored.
- The address is extended to bit[63] by copying bit[55].

Note

Any sign extension required by SMMU_CBn_TCR2.SEP is not performed by the SMMU. The SMMU only sign-extends from bit[55].

SMMU_CBn_TLBIVA bit assignments, AArch32 translation scheme

The SMMU_CBn_TLBIVA bit assignments for an AArch32 translation scheme are:



VA, bits[31:12]

Virtual Address, the input address to the invalidation operation.

Bits[11:8] Reserved.

ASID, bits[7:0]

Address Space Identifier, the input to the invalidate operation.

This field is reserved in HYPC and MONC banks and must be written as zero.

16.5.33 SMMU_CB*n*_TLBIVAA, TLB Invalidate by VA All ASID

The SMMU_CB*n*_TLBIVAA characteristics are:

Purpose Invalidates all of the unlocked TLB entries that match the VA provided as an argument, regardless of the ASID. This operation includes global entries if appropriate.

Usage constraints For Non-secure, non-HYPC context banks, this operation only has to apply to unlocked TLB entries associated with the VMID used for the stage 1 translation context bank.

If SMMU_CBARn.HYPC has the value 1, this operation is UNPREDICTABLE.

If SMMU_CBA2Rn.MONC is 1, this operation is UNPREDICTABLE.

In an implementation that supports two Security states, this register must operate on all unlocked TLB entries associated with the Security state of the stage 1 translation context bank. The VMID of Secure context banks is ignored for TLB matching purposes, even in implementations that support a VMID for Secure banks.

See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

There are two register formats for this register, AArch32 or AArch64, dependent on SMMU_CBA2Rn.VA64:

- If SMMU_CBA2Rn.VA64 is one, then AArch64 format is selected. The programmer should use 64 bit accesses to this register. If 32-bit accesses are used then writes to the top 32 bits are ignored and writes to the lower 32 bits are zero extended.
- If SMMU_CBA2Rn.VA64 is zero, then AArch32 format is selected. The programmer should use 32 bit accesses to this register. If 64-bit accesses are used then the top 32 bits are ignored.

See *TLB maintenance operations on page 5-137* for more details about the usage model.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

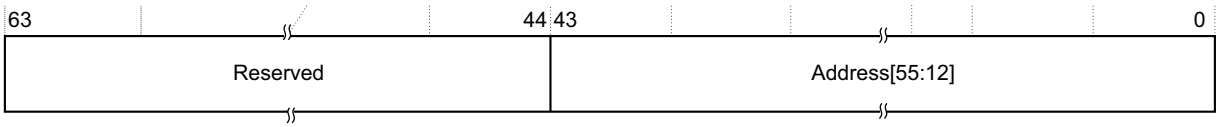
Attributes In AArch64, a 64-bit WO register.

In AArch32, a 32-bit WO register.

Reads are SBZ. See also Table 16-1 on page 16-286.

SMMU_CBn_TLBIVAA bit assignments, AArch64 translation scheme

The SMMU_CB*n*_TLBIVAA bit assignments for an AArch64 translation scheme are:



Bits[63:44] Reserved.

Address[55:12], bits[43:0]

Bits[55:12] of the address to be invalidated:

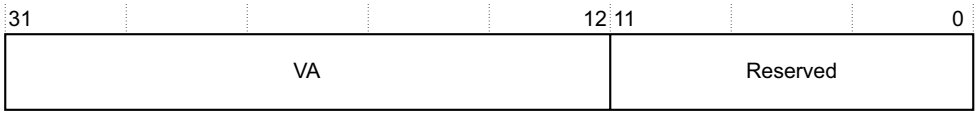
- If the translation granule is 64KB, then bits corresponding to Address[15:12] are ignored.
- If the translation granule is 16KB, then bits corresponding to Address[13:12] are ignored.
- The address is extended to bit[63] by copying bit[55].

Note

Any sign extension required by SMMU_CBn_TCR2.SEP is not performed by the SMMU. The SMMU only sign-extends from bit[55].

SMMU_CBn_TLBIVAA bit assignments, AArch32 translation scheme

The SMMU_CBn_TLBIVAA bit assignments for an AArch32 translation scheme are:



VA, bits[31:12]

Virtual Address, the input address to the invalidation operation.

Bits[11:0] Reserved.

16.5.34 SMMU_CB*n*_TLBIVAAL, TLB Invalidate by VA, All ASID, Last level

The SMMU_CB*n*_TLBIVAAL characteristics are:

Purpose Invalidates all of the unlocked TLB entries that match the VA provided as an argument, regardless of the ASID, that hold information from the final level of lookup. This operation includes global entries if appropriate.

Usage constraints This operation is only required to invalidate TLB entries that include information from the final level of lookup.

For Non-secure, non-HYPC context banks the operation only has to apply to unlocked TLB entries associated with the VMID used for a stage 1 translation context bank. The ASID is not checked for global entries in the TLB.

If SMMU_CBARn.HYPC has the value 1, this operation is UNPREDICTABLE.

If SMMU_CBA2Rn.MONC is 1, this operation is UNPREDICTABLE.

In an implementation that supports two Security states, this register must operate on all matching unlocked TLB entries associated with the Security state of the stage 1 translation context bank. The VMID of Secure context banks is ignored for TLB matching purposes, even in implementations that support a VMID for Secure banks.

See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

There are two register formats for this register, AArch32 or AArch64, dependent on SMMU_CBA2Rn.VA64:

- If SMMU_CBA2Rn.VA64 is one, then AArch64 format is selected. The programmer should use 64 bit accesses to this register. If 32-bit accesses are used then writes to the top 32 bits are ignored and writes to the lower 32 bits are zero extended.
- If SMMU_CBA2Rn.VA64 is zero, then AArch32 format is selected. The programmer should use 32 bit accesses to this register. If 64-bit accesses are used then the top 32 bits are ignored.

See *TLB maintenance operations on page 5-137* for more details about the usage model.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS.

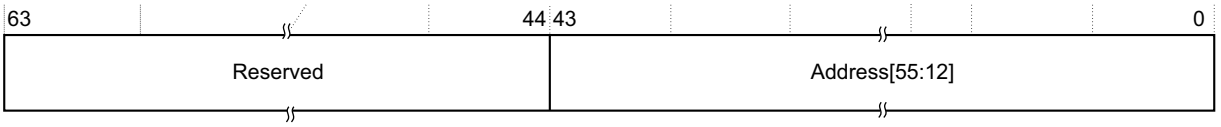
Attributes In AArch64, a 64-bit WO register.

In AArch32, a 32-bit WO register.

Reads are SBZ. See also Table 16-1 on page 16-286.

SMMU_CBn_TLBIVAAL bit assignments, AArch64 translation scheme

The SMMU_CB*n*_TLBIVAAL bit assignments for an AArch64 translation scheme are:



Bits[63:44] Reserved.

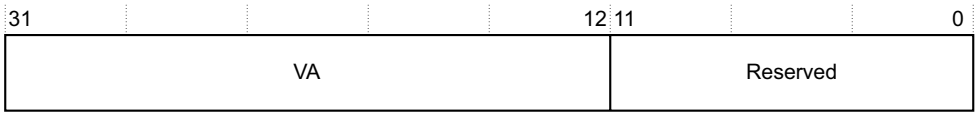
Address[55:12], bits[43:0]

Bits[55:12] of the address to be invalidated:

- If the translation granule is 64KB, then bits corresponding to Address[15:12] are ignored.
- If the translation granule is 16KB, then bits corresponding to Address[13:12] are ignored.
- The address is extended to bit[63] by copying bit[55].

SMMU_CBn_TLBIVAAL bit assignments, AArch32 translation scheme

The SMMU_CBn_TLBIVAAL bit assignments for an AArch32 translation scheme are:



VA, bits[31:12]

Virtual Address, the input address to the invalidation operation.

Bits[11:0] Reserved.

16.5.35 SMMU_CB*n*_TLBIVAL, TLB Invalidate by VA, Last level

The SMMU_CB*n*_TLBIVAL characteristics are:

Purpose Invalidates all of the unlocked TLB entries that match the VA and ASID provided as arguments, that hold the information from the final level of lookup.

Usage constraints This operation is only required to invalidate TLB entries that include information from the final level of lookup.

For Non-secure, non-HYPC context banks, the operation only has to apply to TLB entries associated with the VMID used for a stage 1 translation context bank. The ASID is not checked for global entries in the TLB.

If SMMU_CBARn.HYPC has the value 1, this operation only has to apply to TLB entries associated with hypervisor contexts. VMID and ASID are therefore irrelevant to the operation. See *Hypervisor contexts (HYPC) on page 2-86* for more information.

In an implementation that supports two Security states, this register must operate on all matching unlocked TLB entries associated with the Security state of the stage 1 translation context bank. The VMID of Secure context banks is ignored for TLB matching purposes, even in implementations that support a VMID for Secure banks.

See *The Context Bank Attribute Register, SMMU_CBARn on page 2-78* for more information.

There are two register formats for this register, AArch32 or AArch64, dependent on SMMU_CBA2Rn.VA64:

- If SMMU_CBA2Rn.VA64 is one, then AArch64 format is selected. The programmer should use 64 bit accesses to this register. If 32-bit accesses are used then writes to the top 32 bits are ignored and writes to the lower 32 bits are zero extended.
- If SMMU_CBA2Rn.VA64 is zero, then AArch32 format is selected. The programmer should use 32 bit accesses to this register. If 64-bit accesses are used then the top 32 bits are ignored.

See *TLB maintenance operations on page 5-137* for more details about the usage model.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

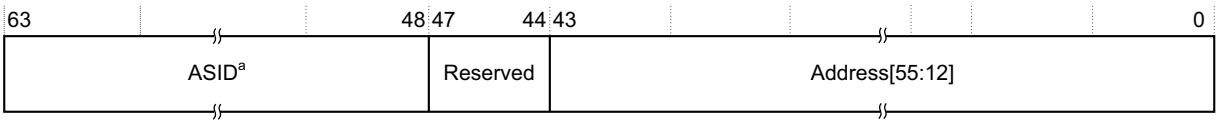
Attributes In AArch64, a 64-bit WO register.

In AArch32, a 32-bit WO register.

Reads are SBZ. See also Table 16-1 on page 16-286.

SMMU_CBn_TLBIVAL bit assignments, AArch64 translation scheme

The SMMU_CB*n*_TLBIVAL bit assignments for an AArch64 translation scheme are:



a. For operations that do not require an ASID, this field is reserved.

ASID, bits[63:48]

Address Space Identifier, the input to the invalidation operation.

When SMMU_CBn_TCR2.AS==0, the upper 8 bits of this field must be written as 0.

This field is reserved in HYPC and MONC banks and must be written as zero.

Bits[47:44] Reserved.

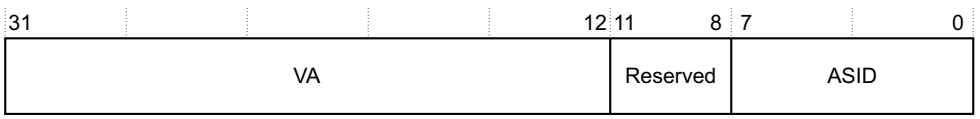
Address[55:12], bits[43:0]

Bits[55:12] of the address to be invalidated:

- If the translation granule is 64KB, then bits corresponding to Address[15:12] are ignored.
- If the translation granule is 16KB, then bits corresponding to Address[13:12] are ignored.
- The address is extended to bit[63] by copying bit[55].

SMMU_CBn_TLBIVAL bit assignments, AArch32 translation scheme

The SMMU_CBn_TLBIVAL bit assignments for an AArch32 translation scheme are:



VA, bits[31:12]

Virtual Address, the input address to the invalidation operation.

Bits[11:8] Reserved.

ASID, bits[7:0]

Address Space Identifier, the input to the invalidate operation.

16.5.36 SMMU_CB*n*_TLBSTATUS, TLB Status

The SMMU_CB*n*_TLBSTATUS characteristics are:

Purpose Indicates the status of any TLB maintenance operations issued before the most recent SMMU_CBn_TLBSYNC operation.

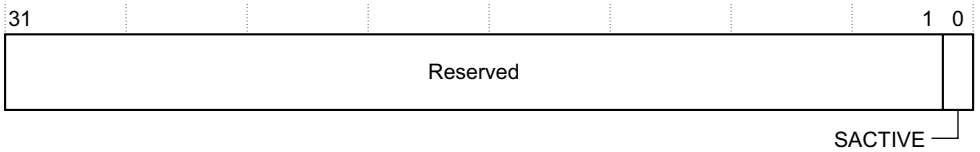
See *TLB maintenance operations on page 5-137* for more details about the usage model.

Usage constraints No usage constraints apply.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A 32-bit RO register that resets to 0. Writes are ignored. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_STATUS bit assignments are:



Bits[31:1] Reserved.

SACTIVE, bit[0]

SMMU_CBn_TLBSYNC operation active.

The possible values of this bit are:

0 All TLB invalidate operations issued before the most recent SMMU_CBn_TLBSYNC operation have completed.

1 Some TLB invalidate operations issued before the most recent SMMU_CBn_TLBSYNC operation have not completed.

16.5.37 SMMU_CB*n*_TLBSYNC, TLB Synchronize Invalidate

The SMMU_CB*n*_TLBSYNC characteristics are:

Purpose Initiates a synchronization operation that ensures the completion of any TLB invalidate operations previously accepted in the corresponding translation context bank.

Usage constraints This operation operates in the scope of the translation context bank it resides in. After being accepted, the operation does not complete until all translation context bank TLB invalidate operations accepted by the SMMU before the synchronize operation was accepted are complete.

See *TLB maintenance operations on page 5-137* for more details about the usage model.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS.

Attributes A 32-bit WO register. Reads are SBZ. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_TLBSYNC bit assignments are reserved.

16.5.38 SMMU_CB*n*_TCR, Translation Control Register

The SMMU_CB*n*_TCR characteristics are:

Purpose Determines translation properties, including which of the Translation Table Base Registers, SMMU_CBn_TTBRm, defines the base address for the translation table walk required when an input address is not found in the TLB.

Usage constraints No usage constraints apply.

Configurations The format of this register depends on the value of SMMU_CBA2Rn.VA64 and SMMU_CB*n*_TCR.EAE. See *Multi-format registers and reserved fields on page 16-293*.

Note

This document refers to the *effective value* of a register bit. The *effective value* of the SMMU_CB*n*_TCR.EAE bit is 1 when any of the following apply:

- The SMMU_CBn_TCR.EAE bit is set to 1.
- The transaction is subject to stage 2 translation.
- The translation uses the AArch64 translation scheme.
- The translation uses a HYPC bank.

In all other cases the effective value of SMMU_CBn_TCR.EAE is 0.

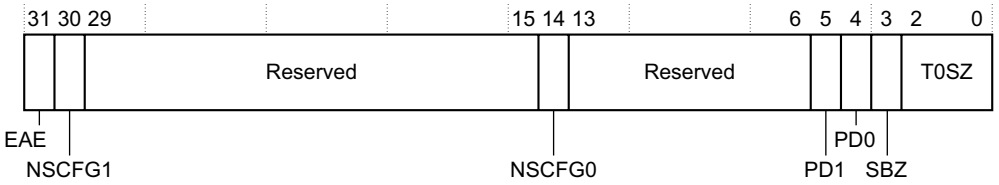
It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

In SMMUv1, this register is SMMU_CBn_TTBCR.

Attributes A 32-bit RW register with an UNKNOWN reset value. See also Table 16-1 on page 16-286.

Register format when SMMU_CB*n*_CBA2R.VA64 is 0 and SMMU_CB*n*_TCR.EAE is 0

The SMMU_CB*n*_TCR bit assignments in this format are:



EAE, bit[31] Extended Address Enable.

The possible values of this bit are:

0 Use the translation system defined in the AArch32 Short-descriptor.

1 Use the translation system defined in the AArch32 Long-descriptor.

NSCFG1, bit[30]

Non-secure attribute for the memory associated with a translation table walk using SMMU_CB*n*_TTBR1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* for more information.

This field only applies to a Secure translation context bank. Otherwise, it is ignored.

Bits[29:15] Reserved.

NSCFG0, bit[14]

Non-secure attribute for the memory associated with a translation table walk using SMMU_CB*n*_TTBR0. This field only applies to a Secure translation context bank. Otherwise, it is ignored. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339*.

Bits[13:6] Reserved.

PD1, bit[5] Translation table walk disable for translations using TTBR1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1. The encoding of this bit is:

0 Perform translation table walks using TTBR1.

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

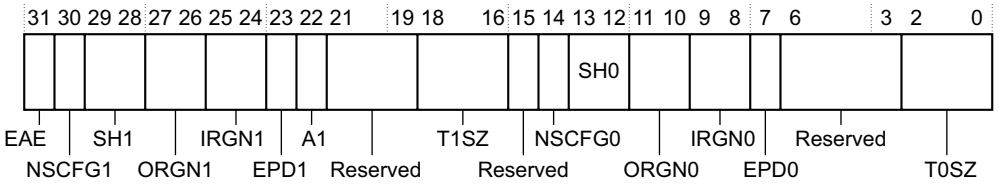
PD0, bit[4] Translation table walk disable for translations using TTBR0. This bit controls whether a translation table walk is performed on a TLB miss for an address that is translated using TTBR0. The meanings of the possible values of this bit are equivalent to those for the PD1 bit.

SBZ, bit[3] Should-be-Zero.

T0SZ, bits[2:0] The size offset of the SMMU_CB*n*_TTBR0 addressed region. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *Defining the VA subranges for stage 1 translations on page 1-30*.

Register format when SMMU_CB*n*_CBA2R.VA64 is 0 and SMMU_CB*n*_TCR.EAE is 1

The SMMU_CB*n*_TCR bit assignments in this format are:



EAE Extended Address Enable.

The possible values of this bit are:

0 Use the translation system defined in the Short-descriptor

1 Use the translation system defined in the Long-descriptor.

Note

This bit is ignored and treated as 1 for stage 2 translation contexts and for Hypervisor contexts that use the AArch32 Long-descriptor translation scheme.

NSCFG1, bit[30]

Non-secure attribute for the memory associated with a translation table walk using SMMU_CB*n*_TTBR1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* for more information.

This field only applies to a Secure translation context bank. Otherwise, it is ignored.

SH1, bits [29:28]

Shareable attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *SH1, SH0 encoding on page 16-332* for more information.

ORGN1, bits [27:26]

Outer Cacheability Attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *ORGN1, ORGN0, IRGN1, IRGN0 encoding on page 16-332* for more information.

IRGN1, bits [25:24]

Inner Cacheability Attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *ORGN1, ORGN0, IRGN1, IRGN0 encoding on page 16-332* for more information.

EPD1, bit[23] Translation Walk Disable for the SMMU_CB*n*_TTBR1 region where SMMU_CB*n*_TCR.EAE==1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* for more information. Controls whether a translation table walk is performed on a TLB miss when SMMU_CB*n*_TTBR1 is used.

This bit is UNK/SBZ if either:

- SMMU_CB*n*_TCR.EAE==0.
- SMMU_CBARn.HYPC==1.

The possible values of this bit are:

0 If a TLB miss occurs when SMMU_CB*n*_TTBR1 is used, a translation table walk is performed.

1 If a TLB miss occurs when SMMU_CB*n*_TTBR1 is used, no translation table walk is performed and an L1 Section translation fault is returned.

This function is the same as PD1. The bit position is moved depending on the setting of SMMU_CB*n*_TCR.EAE.

A1, bit[22] Select the ASID from the SMMU_CB*n*_TTBR1 or SMMU_CB*n*_TTBR0 ASID field. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* for more information.

The possible values of this bit are:

0 Select the ASID from the SMMU_CB*n*_TTBR0 ASID field.

1 Select the ASID from the SMMU_CB*n*_TTBR1 ASID field.

Bits[21:19] Reserved.

T1SZ, bits[18:16]

The size offset of the SMMU_CB*n*_TTBR1 addressed region. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *Defining the VA subranges for stage 1 translations on page 1-30*.

Bit[15] Reserved.

NSCFG0, bit[14]

Non-secure attribute for the memory associated with a translation table walk using SMMU_CB*n*_TTBR0. This field only applies to a Secure translation context bank. Otherwise, it is ignored. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339*.

SH0, bits[13:12]

Shareable attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR0. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *SH1, SH0 encoding on page 16-332* for more information.

ORGN0, bits[11:10]

Outer Cacheability Attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR0. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *ORGN1, ORGN0, IRGN1, IRGN0 encoding on page 16-332* for more information.

IRGN0, bits[9:8]

Inner Cacheability Attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR0. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *ORGN1, ORGN0, IRGN1, IRGN0 encoding on page 16-332* for more information.

EPD0, bit[7] Translation Walk Disable for the SMMU_CB*n*_TTBR0 region, where EAE==1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* for more information. Controls whether a translation table walk is performed on a TLB miss when SMMU_CB*n*_TTBR0 is used.

This bit is UNK/SBZ if either:

- SMMU_CB*n*_TCR.EAE==0.

- SMMU_CBARn.HYPC==1.

The possible values of this bit are:

0 If a TLB miss occurs when SMMU_CB*n*_TTBR0 is used, a translation table walk is performed.

1 If a TLB miss occurs when SMMU_CB*n*_TTBR0 is used, no translation table walk is performed and an L1 Section translation fault is returned.

This function is the same as PD0.

Bits[6:3] Reserved.

T0SZ, bits[2:0]

The size offset of the SMMU_CB*n*_TTBR0 addressed region. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *Defining the VA subranges for stage 1 translations on page 1-30*.

SH1, SH0 encoding

Table 16-6 shows the encoding of the SH1 and SH0 fields.

Table 16-6 SMMU_CB*n*_TCR.SH1 & SH0 encoding

SH[1]	SH[0]	Normal memory
0	0	Non-shareable
0	1	UNPREDICTABLE
1	0	Outer Shareable
1	1	Inner Shareable

ORGN1, ORGN0, IRGN1, IRGN0 encoding

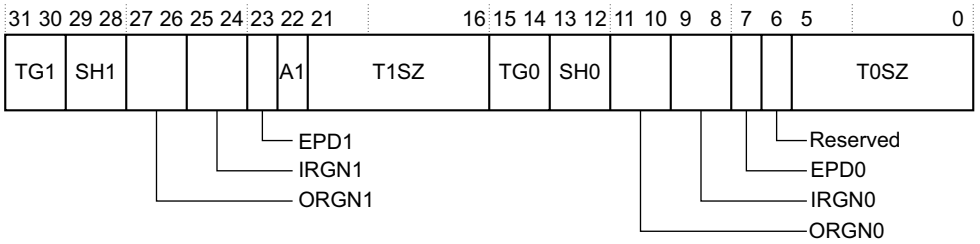
Table 16-7 shows the encoding of the ORGN1, ORGN0, IRGN1 and IRGN0 fields.

Table 16-7 SMMU_CB*n*_TCR.ORGN1, ORGN0, IRGN1, IRGN0 encoding

RGN[1]	RGN[0]	Description
0	0	Non-cacheable Normal memory
0	1	Write-Back, Write-Allocate cacheable
1	0	Write-Through cacheable
1	1	Write-Back, no Write-Allocate cacheable

Register format when SMMU_CB*n*_CBA2R.VA64 is 1

When the SMMU_CBA2Rn.VA64 bit is set to 1, the SMMU uses the AArch64 translation scheme. The SMMU_CBn_TCR bit assignments for the AArch64 translation scheme are:



TG1, bits[31:30]

The page granule size for TTBR1, for non-HYPC, non-MONC stage 1 translations. The possible values of this field are:

`0b00` Reserved.

`0b01` 16KB granule size.

`0b10` 4KB granule size.

`0b11` 64KB granule size.

If software sets this field to a reserved value, or a size that has not been implemented, hardware treats the field as if it is programmed to an IMPLEMENTATION DEFINED choice of the sizes that are implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

SH1, bits [29:28]

Shareable attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *SH1, SH0 encoding on page 16-332* for more information.

ORGN1, bits [27:26]

Outer Cacheability Attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *ORGN1, ORGN0, IRGN1, IRGN0 encoding on page 16-332* for more information.

IRGN1, bits [25:24]

Inner Cacheability Attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *ORGN1, ORGN0, IRGN1, IRGN0 encoding on page 16-332* for more information.

EPD1, bit[23] Translation Walk Disable for the SMMU_CB*n*_TTBR1 region. Controls whether a translation table walk is performed on a TLB miss when SMMU_CB*n*_TTBR1 is used.

This bit is UNK/SBZ if either:

- SMMU_CB*n*_TCR.EAE==0.
- SMMU_CBARn.HYPC==1.

The possible values of this bit are:

0 If a TLB miss occurs when SMMU_CB*n*_TTBR1 is used, a translation table walk is performed.

1 If a TLB miss occurs when SMMU_CB*n*_TTBR1 is used, no translation table walk is performed and an L1 Section translation fault is returned.

This function is the same as PD1. The bit position is moved depending on the setting of SMMU_CB*n*_TCR.EAE.

A1, bit[22] Select the ASID from the SMMU_CB*n*_TTBR1 or SMMU_CB*n*_TTBR0 ASID field. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* for more information.

The possible values of this bit are:

0 Select the ASID from the SMMU_CB*n*_TTBR0 ASID field.

1 Select the ASID from the SMMU_CB*n*_TTBR1 ASID field.

T1SZ, bits[21:16]

The size offset of the SMMU_CB*n*_TTBR1 addressed region, encoded as a 6-bit unsigned number, giving the size of the region as $2^{64-T1SZ}$. See *Defining the VA subranges for stage 1 translations on page 1-30*.

TG0, bits[15:14]

The page granule size for TTBR0. The possible values of this field are:

`0b00` 4KB granule size.

`0b01` 64KB granule size.

`0b10` 16KB granule size.

`0b11` Reserved.

If software sets this field to a reserved value, or a size that has not been implemented, hardware treats the field as if it is programmed to an IMPLEMENTATION DEFINED choice of the sizes that are implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

SH0, bits[13:12]

Shareable attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR0. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *SH1, SH0 encoding on page 16-332* for more information.

ORGN0, bits[11:10]

Outer Cacheability Attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR0. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *ORGN1, ORGN0, IRGN1, IRGN0 encoding on page 16-332* for more information.

IRGN0, bits[9:8]

Inner Cacheability Attributes for the memory associated with the translation table walks using SMMU_CB*n*_TTBR0. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* and *ORGN1, ORGN0, IRGN1, IRGN0 encoding on page 16-332* for more information.

EPD0, bit[7] Translation Walk Disable for the SMMU_CB*n*_TTBR0 region, where EAE==1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* for more information. Controls whether a translation table walk is performed on a TLB miss when SMMU_CB*n*_TTBR0 is used.

This bit is UNK/SBZ if either:

- SMMU_CB*n*_TCR.EAE==0.
- SMMU_CBARn.HYPC==1.

The possible values of this bit are:

0 If a TLB miss occurs when SMMU_CB*n*_TTBR0 is used, a translation table walk is performed.

1 If a TLB miss occurs when SMMU_CB*n*_TTBR0 is used, no translation table walk is performed and an L1 Section translation fault is returned.

This function is the same as PD0. The bit position is moved depending on the setting of SMMU_CB*n*_TCR.EAE.

Bit[6] Reserved.

T0SZ, bits[5:0]

The size offset of the SMMU_CB*n*_TTBR0 addressed region, encoded as a 6-bit unsigned number, giving the size of the region as $2^{64-T0SZ}$. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339*, and *Defining the VA subranges for stage 1 translations on page 1-30*.

16.5.39 SMMU_CB*n*_TCR2, Translation Control Register 2

The SMMU_CB*n*_TCR2 characteristics are:

Purpose Extends SMMU_CBn_TCR by adding control information about the translation granule size and the size of the intermediate physical address.

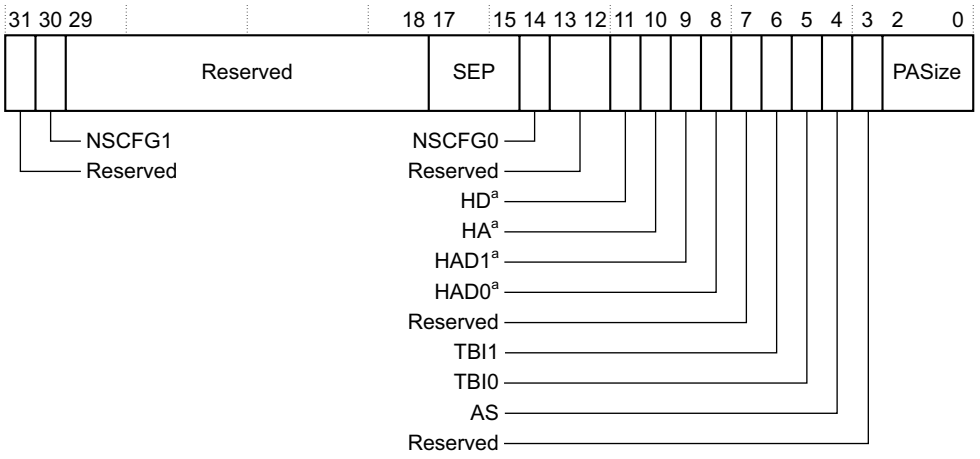
Usage constraints No usage constraints apply.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

This register is not provided in SMMUv1.

Attributes A 32-bit RW register with an UNKNOWN reset value. See also Table 16-1 on page 16-286.

The SMMU_CB*n*_TCR2 bit assignments are:



a. Reserved in SMMUv1

Bit[31] Reserved.

NSCFG1, bit[30]

Non-secure attribute for the memory associated with a translation table walk using SMMU_CB*n*_TTBR1. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* for more information.

This field only applies to a Secure translation context bank. Otherwise, it is ignored.

Bits[29:18] Reserved.

SEP, bits[17:15]

Sign Extension Position. The bit position from which to sign-extend the stage 1 input address when required.

The encoding of this field is:

`0b000` Bit[31]

`0b001` Bit[35]

`0b010` Bit[39]

`0b011` Bit[41]

`0b100` Bit[43]

`0b101` Bit[47]

`0b110` Reserved. This bit must be treated as `0b111`.

`0b111` The meaning of this bit depends on the upstream address bus size indicated by SMMU_IDR2.UBS, as follows:

UBS==0b1111, 64-bit bus

No sign extension is performed.

UBS==0b0101, 49 bit bus

Bit[48] is used as the sign bit, and is replicated to bits[63:49].

Other valid encodings of UBS

The input address is zero-extended.

If the value of this bit changes for a context bank, software must invalidate any affected TLB entries.

Note

- This field is ignored for the AArch32 Long-descriptor and AArch32 Short-descriptor translation schemes. If SEP specifies a bit above that implemented on the upstream input address bus, addresses are zero-extended. Sign-extension does not apply to address translation operations.
- This field is ignored for HYPC and MONC banks.

NSCFG0, bit[14]

Non-secure attribute for the memory associated with a translation table walk using SMMU_CB*n*_TTBR0. This field only applies to a Secure translation context bank. Otherwise, it is ignored. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339*.

Bits[13:12] Reserved.

HD, bit[11], SMMUv2 only

Hardware management of Dirty bit. The possible values of this bit are:

0 Stage 1 Dirty bit update disabled.

1 Stage 1 Dirty bit update enabled if the value of the HA bit 1.

If the value of SMMU_IDR1.HAFDBS is 0, this bit is RES0.

When implemented as an R/W bit, this bit is not used by AArch32 translation contexts, meaning it is ignored when SMMU_CBA2Rn.VA64 is 0.

HA, bit[10], SMMUv2 only

Hardware management of Access flag. The possible values of this bit are:

0 Stage 1 Access flag update disabled.

1 Stage 1 Access flag update enabled.

If the value of SMMU_IDR1.HAFDBS is 0, this bit is RES0.

When implemented as an R/W bit, this bit is not used by AArch32 translation contexts, meaning it is ignored when SMMU_CBA2Rn.VA64 is 0.

HAD1, bit[9], SMMUv2 only

Hierarchical Attribute Disable for the TTBR1 region. The possible values of this bit are:

0 Hierarchical Attributes are enabled.

1 Hierarchical Attributes are disabled.

This bit is not used by AArch32 translation contexts, meaning it is ignored when SMMU_CBA2Rn.VA64 is 0. It is not used and is ignored by AArch64 HYPC, and MONC contexts.

This bit is RES0 if SMMU_IDR2.HADS == 0.

HAD0, bit[8], SMMUv2 only

Hierarchical Attribute Disable for the TTBR0 region. The possible values of this bit are:

0 Hierarchical Attributes are enabled.

1 Hierarchical Attributes are disabled.

This bit is not used by AArch32 translation contexts, meaning it is ignored when SMMU_CBA2Rn.VA64 is 0.

This bit is RES0 if SMMU_IDR2.HADS == 0.

Bits[11:8], SMMUv1

Reserved, RES0.

Bit[7] Reserved.

TBI1, bit[6] Top Byte Ignored. Indicates whether the top byte of the input address can be used for address match for the TTBR1 region. Possible values for this bit are:

0 Top byte is used in the address calculation.

1 Top byte is ignored in the address calculation.

Note

This bit affects only address translation operations that might supply a full 64-bit tagged address.

TBI0, bit[5] Top Byte Ignored. Indicates whether the top byte of the input address can be used for address match for the TTBR0 region. Possible values for this bit are:

0 Top byte is used in the address calculation.

1 Top byte is ignored in the address calculation.

Note

This bit affects only address translation operations that might supply a full 64-bit tagged address.

AS, bit[4] ASID Size. Possible values for this field are:

0 8-bit.

1 16-bit.

Note

8-bit ASIDs extend to 16-bits for all TLB matching purposes. When this bit is set to 0, the upper 8 bits of SMMU_CB*n*_TTBR0 and SMMU_CB*n*_TTBR1 are:

- Ignored by hardware, except when reading back the register.
- Treated as 0 when used for allocation and for TLB matching.

Bit[3] Reserved.

PASize, bits[2:0]

Physical Address Size. If the SMMU supports a larger address space than a device provides, this field enables software to reduce the amount of SMMU address space.

The encoding of this field is:

`0b000` 32-bits, 4GB

`0b001` 36-bits, 64GB

`0b010` 40-bits, 1TB

`0b011` 42-bits, 4TB

`0b100` 44-bits, 16TB

`0b101` 48-bits, 256TB

All other values are reserved, and treated as 48-bits.

16.5.40 SMMU_CB*n*_TTBR*m*, Translation Table Base Registers

The SMMU_CB*n*_TTBR1 and SMMU_CB*n*_TTBR0 characteristics are:

Purpose Holds the translation table base addresses:
- SMMU_CB*n*_TTBR1 holds the base address of translation table 1.
- SMMU_CB*n*_TTBR0 holds the base address of translation table 0.

Usage constraints Depending on the translation scheme selected, either an address size fault or a translation fault is generated if an IPA exceeds the number of bits that the SMMU_IDR2.OAS field defines.

Configurations The format of this register depends on the translation scheme selected:
- *AArch32 Short-descriptor translation scheme*.
- *AArch32 Long-descriptor translation schemes on page 16-340*.

See also *Multi-format registers and reserved fields on page 16-293*.

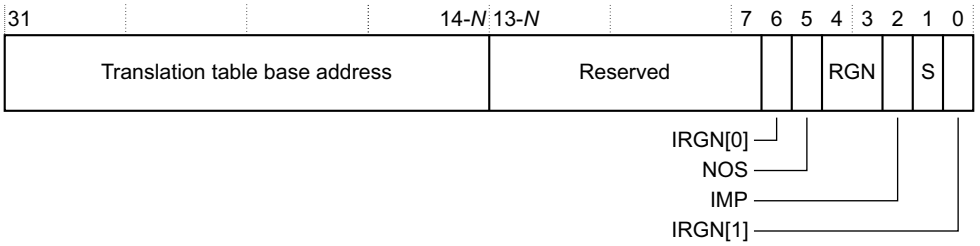
It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes A RW register with an UNKNOWN reset value. When using the AArch32 Short-descriptor translation table format, this is a 32-bit register. Otherwise, it is a 64-bit register. See also Table 16-1 on page 16-286

The format of the register depends on the current translation scheme:
- *AArch32 Short-descriptor translation scheme*.
- *AArch32 Long-descriptor translation schemes on page 16-340*.
- *AArch64 translation scheme on page 16-341*.

AArch32 Short-descriptor translation scheme

When using the AArch32 Short-descriptor translation scheme, the SMMU_CBn_TTBRm bit assignments are:



Translation table base address, bits[31:14-*N*] Translation table base 1 address or translation table base 0 address, bits [47:14-*N*], where *N* determines the required alignment of the translation table, which must be aligned to 2^{14-N} bytes:
- For SMMU_CB*n*_TTBR1, *N*==0.
- For SMMU_CB*n*_TTBR0, *N* is determined by SMMU_CBn_TCR.T0SZ.

Bits[13-N:7] Reserved.

IRGN[0], bit[6]

Inner Region bits. See also SMMU_CB*n*_TTBR*m*.IRGN[1].

These bits indicate the Inner cacheability attributes for the memory associated with the translation table walks.

The encoding of bits[0,6] is:

`0b00` Inner Non-cacheable Normal memory.

`0b01` Inner Write-Back Write-Allocate cacheable Normal memory.

`0b10` Inner Write-Through cacheable Normal memory.

`0b11` Inner Write-Back no Write-Allocate cacheable Normal memory.

NOS, bit[5] Not Outer Shareable bit.

This bit indicates the Outer Shareable attribute for the memory associated with a translation table walk that has the Shareable attribute, indicated by SMMU_CB*n*_TTBR1.S or SMMU_CB*n*_TTBR0.S having the value 1.

The possible values of this bit are:

0 Outer Shareable.

1 Inner Shareable.

RGN, bits[4:3] Region bits, indicate the Outer cacheable attributes for the memory associated with the translation table walk.

The encoding of this field is:

`0b00` Non-cacheable Normal memory.

`0b01` Outer Write-Back Write-Allocate cacheable.

`0b10` Outer Write-Through cacheable.

`0b11` Outer Write-Back no Write-Allocate cacheable.

IMP, bit[2] IMPLEMENTATION DEFINED bit.

The effect of this bit is IMPLEMENTATION DEFINED. If the translation table implementation does not include any IMPLEMENTATION DEFINED features, this bit is SBZ.

S, bit[1] Shareable bit, indicates the Shareable attribute for the memory associated with a translation table walk.

The possible values of this bit are:

0 Non-shareable.

1 Shareable.

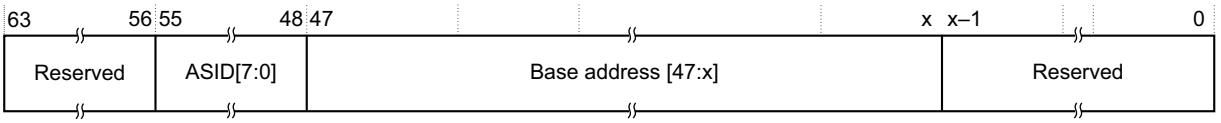
IRGN[1], bit[0]

Inner region bits. See also SMMU_CBn_TTBR*m*.IRGN[0].

These bits indicate the Inner cacheability attributes for the memory associated with a translation table walk.

AArch32 Long-descriptor translation schemes

When using the AArch32 Long-descriptor translation scheme, the SMMU_CBn_TTBRm assignments are:



Bits[63:56] Reserved.

ASID[7:0], bits[55:48]

The Address Space Identifier associated with this base address.

SMMU_CBn_TCR.A1 determines the selection between SMMU_CB*n*_TTBR0.ASID and SMMU_CB*n*_TTBR1.ASID.

Base Address [47:x]

Translation table base 1 or 0 address, bits [47:*x*].

The value *x* is determined by the T0SZ or T1SZ field in SMMU_CBn_TCR, according to the algorithm specified in *The algorithm for finding the translation table descriptors on page 2-79*.

The number of implemented bits is determined by the maximum address sizes, as defined by the SMMU_IDR2.IAS and SMMU_IDR2.OAS fields. Unimplemented bits are RES0.

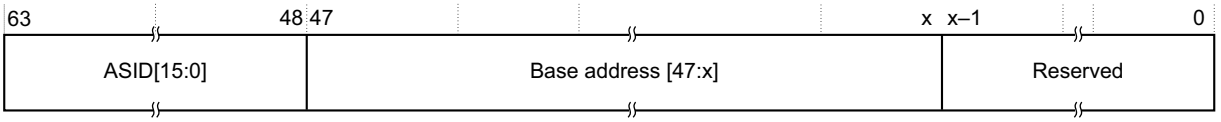
Note

AArch32 state does not support addresses larger than 40 bits, therefore bits[47:40] are always RES0.

Bits[*x*-1:0] Reserved, SBZ. Behavior is UNPREDICTABLE if bits[2:0] are not `0b000`

AArch64 translation scheme

When using the AArch64 Long-descriptor translation scheme, the SMMU_CBn_TTBRm bit assignments are:



ASID[15:0], bits[63:48]

The Address Space Identifier associated with this base address.

SMMU_CBn_TCR.A1 determines the selection between SMMU_CB*n*_TTBR0.ASID and SMMU_CB*n*_TTBR1.ASID.

ASID[15:8] is valid only when the SMMU_CBn_TCR2.AS bit is set to 1, otherwise ASID[15:8] is reserved, treated as 0, and must be written as 0.

Base Address [47:x]

Translation table base 1 or 0 address, bits [47:*x*].

The value *x* is determined by several fields, including the T0SZ or T1SZ field in SMMU_CBn_TCR and SMMU_CBn_TCR.SL0, according to the algorithm specified in *The algorithm for finding the translation table descriptors on page 2-79*.

The number of implemented bits is determined by the maximum address sizes, as defined by the SMMU_IDR2.IAS and SMMU_IDR2.OAS fields. Unimplemented bits are RAZ/WI.

Bits[*x*-1:0] Reserved, SBZ. Behavior is UNPREDICTABLE if bits[2:0] are not `0b000`.

Chapter 17
Stage 2 Translation Context Bank Format

This chapter gives the layout of the stage 2 translation context bank. It contains the following sections:

17.1 Stage 1 and stage 2 context bank format differences

The format of a stage 2 translation context bank is similar to a stage 1 translation context bank. The following differences apply to the stage 2 format:

- The following registers do not exist and are UNK/SBZP in the stage 2 translation context bank:
  - SMMU_CB*n*_NMRR.
  - SMMU_CB*n*_PRRR.
  - SMMU_CB*n*_TTBR1.
  - SMMU_CB*n*_CONTEXTIDR.
  - SMMU_CBn_TCR2.
- The following registers do not exist in the stage 2 translation context bank:
  - SMMU_CB*n*_ATSR.
  - SMMU_CB*n*_MAIR1, SMMU_CB*n*_MAIR0.
  - SMMU_CB*n*_PAR.
  - SMMU_CB*n*_ATS1PR.
  - SMMU_CB*n*_ATS1PW.
  - SMMU_CB*n*_ATS1UR.
  - SMMU_CB*n*_ATS1UW.
  - SMMU_CB*n*_TLBIALL.
  - SMMU_CB*n*_TLBIASID.
  - SMMU_CB*n*_TLBIVA.
  - SMMU_CB*n*_TLBIVAA.
  - SMMU_CBn_TLBIVAAL.
  - SMMU_CBn_TLBIVAL.
- The following registers exist only in the stage 2 translation context bank:
  - SMMU_CBn_IPAFAR.
  - SMMU_CBn_TLBIIPAS2.
  - SMMU_CBn_TLBIIPAS2L.
- A stage 2 translation context bank only supports the LPAE mode of operation. This means that SMMU_CBn_TCR.EAE has a fixed value of 1.
- The fields in SMMU_CBn_TCR that relate to SMMU_CB*n*_TTBR1 are UNK/SBZP. See *SMMU_CBn_TTBRm, Translation Table Base Registers on page 16-339* for more information.

17.2 Stage 2 translation context bank address space

The stage 2 translation context bank provides registers for initial translation in implementations that support stage 2 translation.

The SMMU architecture supports 32-bit atomic access to all stage 2 translation context bank format registers, and 64-bit atomic access to the 64-bit registers shown in Table 17-1. Whether 8-bit or 16-bit atomic accesses to these registers are supported is IMPLEMENTATION DEFINED.

Table 17-1 shows the address of the stage 2 translation context bank format registers relative to the offset from the translation context bank base address, SMMU_CBn_BASE. Unless otherwise stated, registers are present in any version of the SMMU architecture.

See also *Stage 1 and stage 2 context bank format differences on page 17-344*.

Table 17-1 Stage 2 translation context bank address space

Offset	Name	Width	Type	Description	Notes
0x00000	SMMU_CBn_SCTLR	32-bit	RW	*SMMU_CBn_SCTLR, System Control Register on page 17-351*	-
0x00004	SMMU_CBn_ACTLR	32-bit	RW	*SMMU_CBn_ACTLR, Auxiliary Control Register on page 16-294*	-
0x00008	SMMU_CBn_RESUME	32-bit	WO	*SMMU_CBn_RESUME, Transaction Resume register on page 16-314*	-
0x0000C-0x0001C	Reserved	-	-	-	-
0x00020	SMMU_CBn_TTBR0	64-bit	RW	*SMMU_CBn_TTBR0, Translation Table Base Register on page 17-360*	-
0x00028-0x0002C	Reserved	-	-	-	-
0x00030	SMMU_CBn_TCR	32-bit	RW	*SMMU_CBn_TCR, Translation Control Register on page 17-358*	-
0x00034-0x0003C	Reserved	-	-	-	-
0x00040-0x00044	Reserved	-	-	Reserved for IMPLEMENTATION DEFINED memory attributes associated with memory encodings. Such attributes are only additional qualifiers to the memory locations, and cannot change the architected behaviors of the memory attributes defined in SMMU_CB*n*_MAIR0 and SMMU_CB*n*_MAIR1. See *SMMU_CBn_MAIRm, Memory Attribute Indirection Registers on page 16-304* for more information.	
0x0004C-0x00054	Reserved	-	-	-	-
0x00058	SMMU_CBn_FSR	32-bit	RW	*SMMU_CBn_FSR, Fault Status Register on page 16-298*	-
0x0005C	SMMU_CBn_FSRRESTORE	32-bit	WO	*SMMU_CBn_FSRRESTORE, Fault Status Restore Register on page 16-301*	-

Table 17-1 Stage 2 translation context bank address space (continued)

Offset	Name	Width	Type	Description	Notes
0x00060	SMMU_CBn_FAR	64-bit	RW	*SMMU_CBn_FAR, Fault Address Register on page 16-298*	-
0x00068	SMMU_CB*n*_FSYNR0	32-bit	RW	*SMMU_CBn_FSYNRn, Fault Syndrome Registers on page 17-348*	-
0x0006C	SMMU_CB*n*_FSYNR1				
0x00070	SMMU_CBn_IPAFAR	64-bit	RW	*SMMU_CBn_IPAFAR, IPA Fault Address Register on page 17-350*	SMMUv2 only
0x00078-0x0062C	Reserved	-	-	-	-
0x00630	SMMU_CBn_TLBIIPAS2	64-bit	WO	*SMMU_CBn_TLBIIPAS2, Invalidate TLB by IPA on page 17-355*	SMMUv2 only 64-bit
0x00638	SMMU_CBn_TLBIIPAS2L	64-bit	WO	*SMMU_CBn_TLBIIPAS2L, Invalidate TLB by IPA, Last level on page 17-356*	SMMUv2 only 64-bit
0x00640-0x00CFC	Reserved	-	-	-	-
0x007F0	SMMU_CBn_TLBSYNC	32-bit	WO	*SMMU_CBn_TLBSYNC, TLB Synchronize Invalidate on page 17-357*	SMMUv2 only
0x007F4	SMMU_CBn_TLBSTATUS	32-bit	RO	*SMMU_CBn_TLBSTATUS, TLB Status on page 17-357*	SMMUv2 only
0x007F8-0x00CFC	Reserved	-	-	-	-
0x00D00-0x00DFC	IMPLEMENTATION DEFINED	-	RW	Reserved for IMPLEMENTATION DEFINED purposes	-
0x00E00-0x00E38	SMMU_CBn_PMEVCNTRm	32-bit	RW	*SMMU_CBn_PMEVCNTRm, Performance Monitors Event Counter registers on page 16-311*	-
0x00E3C-0x00E7C	Reserved	-	-	-	-
0x00E80-0x00EB8	SMMU_CBn_PMEVTYPERm	32-bit	RW	*SMMU_CBn_PMEVTYPERm, Performance Monitors Event Type Registers on page 16-311*	-
0x00EBC-0x00EFC	Reserved	-	-	-	-
0x00F00	SMMU_CBn_PMCFGR	32-bit	RO	*SMMU_CBn_PMCFGR, Performance Monitors Configuration Register on page 16-310*	-
0x00F04	SMMU_CBn_PMCR	32-bit	RW	*SMMU_CBn_PMCR, Performance Monitors Control Register on page 16-311*	-
0x00F08	Reserved	-	-	-	Reserved for PMUSERENR
0x00F0C-0x00F1C	Reserved	-	-	-	-

Table 17-1 Stage 2 translation context bank address space (continued)

Offset	Name	Width	Type	Description	Notes
0x00F20	SMMU_CB*n*_PMCEID0	32-bit	RO	*SMMU_CBn_PMCEIDm, Performance Monitors Common Event Identification registers on page 16-310*	-
0x00F24	SMMU_CB*n*_PMCEID1				
0x00F28-0x00F3C	Reserved	-	-	-	-
0x00F40	SMMU_CBn_PMCNTENSET	32-bit	RW	*SMMU_CBn_PMCNTENSET, Performance Monitors Count Enable Set register on page 16-310*	-
0x00F44	SMMU_CBn_PMCNTENCLR	32-bit	RW	*SMMU_CBn_PMCNTENCLR, Performance Monitors Count Enable Clear register on page 16-310*	-
0x00F48	SMMU_CBn_PMINTENSET	32-bit	RW	*SMMU_CBn_PMINTENSET, Performance Monitors Interrupt Enable Set register on page 16-312*	-
0x00F4C	SMMU_CBn_PMINTENCLR	32-bit	RW	*SMMU_CBn_PMINTENCLR, Performance Monitors Interrupt Enable Clear register on page 16-311*	-
0x00F50	SMMU_CBn_PMOVSCLR	32-bit	RW	*SMMU_CBn_PMOVSCLR, Performance Monitors Overflow Status Clear Register on page 16-312*	-
0x00F54	Reserved	-	-	-	Reserved for PMSWINC
0x00F58	SMMU_CBn_PMOVSSET	32-bit	RW	*SMMU_CBn_PMOVSSET, Performance Monitors Overflow Status Set Register on page 16-312*	-
0x00F5C-0x00FB4	Reserved	-	-	-	-
0x00FB8	SMMU_CBn_PMAUTHSTATUS	32-bit	RO	*SMMU_CBn_PMAUTHSTATUS, Performance Monitors Authentication Status register on page 16-310*	-
0x00FBC-0x00FC8	Reserved	-	-	-	-
0x00FCC	Reserved	-		-	Reserved for PMDEVTYPE
0x00FD0-(PAGESIZE − 0x4)	Reserved	-	-	-	-

17.3 Stage 2 translation context bank register descriptions

This section describes some of the stage 2 translation context bank registers that might be present in an SMMU implementation. The registers described in this section differ slightly to those in the stage 1 translation context bank address space. Unless otherwise stated, registers are present in any version of the SMMU architecture. See Table 17-1 on page 17-345 for the full stage 2 translation context bank address space map.

17.3.1 SMMU_CB*n*_FSYNR*n*, Fault Syndrome Registers

The SMMU_CB*n*_FSYNR1 and SMMU_CB*n*_FSYNR0 characteristics are:

Purpose Holds fault syndrome information about the memory access that caused a synchronous abort exception.

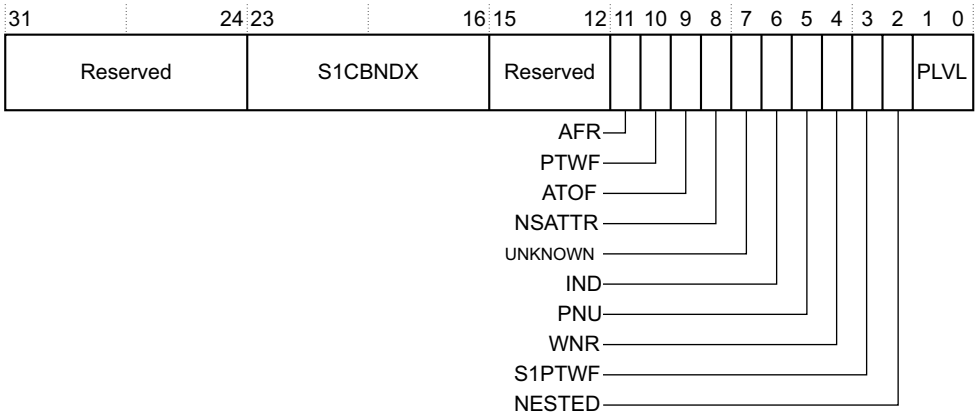
Usage constraints No usage constraints apply.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 1 translation. See SMMU_IDR0.S1TS for more information.

Attributes 32-bit RW registers with UNKNOWN reset values.

SMMU_CB*n*_FSYNR0

The SMMU_CB*n*_FSYNR0 bit assignments are:



Bits[31:24] Reserved.

S1CBNDX[23:16]

Stage 1 Context Bank Index associated with the transaction that caused the fault.

For stage 1 followed by stage 2 translations, this field contains the stage 1 translation context bank index for processing the transaction.

For stage 2 only translation, this field is UNKNOWN.

This field is only present in a stage 2 format translation context bank. In a stage 1 format translation context bank, it is UNK/SBPZ.

This field is only valid if SMMU_IDR0.NTS==1. In an implementation that does not include stage 1 followed by stage 2 translation, this field is UNK/WI.

Note

An SMMU implementation is not required to provide storage for all 8 bits of this field, and might provide only enough bits to represent the highest implemented context bank number.

Bits[15:12] Reserved.

AFR, bit[11] Asynchronous fault recorded. The possible values of this bit are:

0 A fault was recorded synchronously.

1 A fault was recorded asynchronously.

PTWF, bit[10] Page Table Walk Fault. Indicates whether an external fault was recorded during a translation table access. The possible values of this bit are:

0 An external fault did not occur while processing a translation table walk.

1 An external fault occurred while processing a translation table walk.

In SMMUv2, this bit is set to 0 unless an external fault is recorded during a translation table walk.

In SMMUv1, it is UNPREDICTABLE whether this bit is set to 1 for any other fault type.

ATOF, bit[9] Address translation operation fault. The possible values of this bit are:

0 An ATOF fault did not occur.

1 A fault occurred during the processing of an address translation operation.

NSATTR, bit[8]

Non-secure Attribute. The possible values of this bit are:

0 The input transaction has a Secure attribute.

1 The input transaction has a Non-secure attribute.

Bit[7] UNKNOWN and might ignore writes.

IND, bit[6] Instruction Not Data. The possible values of this bit are:

0 Data.

1 Instruction.

PNU, bit[5] Privileged Not Unprivileged. The possible values of this bit are:

0 Unprivileged.

1 Privileged.

WNR, bit[4] Write Not Read. The possible values of this bit are:

0 Read.

1 Write.

Note

For far atomic operations, this field indicates the permission that is missing that prevents the far atomic operation from succeeding. If neither permission is available or any other fault occurs, this field will indicate “Read”.

S1PTWF, bit[3]

A walk fault on a stage 1 translation table access. The possible values of this bit are:

0 A fault did not occur during stage 2 translation of a stage 1 translation table walk.

1 A fault occurred during stage 2 translation of a stage 1 translation table walk.

This field is only valid if the implementation supports stage 1 followed by stage 2 translation. That is, if SMMU_IDR0.NTS==1. In an implementation that does not include stage 1 followed by stage 2 translation, this field is UNK/SBZP.

NESTED, bit[2]

Indicates whether the fault is related to a stage 1 followed by stage 2 translation. The possible values of this bit are:

0 The fault is related to a stage 2 translation.

1 The fault is related to a stage 1 followed by stage 2 translation.

When this bit is set to 0:

- SMMU_CBn_FAR records the IPA that faulted during stage 2.
- SMMU_CBn_IPAFAR also records the IPA that faulted at stage 2.
- SMMU_CBn_FSYNR0.S1CBNDX is UNKNOWN.

When this bit is set to 1:

- SMMU_CBn_FAR records the VA of the requested translation.
- SMMU_CBn_FSYNR0.S1CBNDX indicates the stage 1 context bank that caused the translation.
- SMMU_CBn_IPAFAR records the IPA that faulted at stage 2.

Note

If SMMU_IDR0.NTS==0, this bit is RAZ/WI.

PLVL, bits[1:0]

Translation Table Level, the level in the translation table walk that the fault is associated with. The encoding of this field is:

`0b00` Level 0. SMMUv2 only.

`0b01` Level 1.

`0b10` Level 2.

`0b11` Level 3.

Faults and translation table level association

The translation table level a fault is associated with is:

- For a fault associated with a translation table walk, the level of table walk being performed.
- For a translation fault, the level of translation table that gave the fault. If a disabled translation table walk causes the fault or if the size of the address presented is out of the range specified for matching with any base address register, the fault is reported for level 1.
- For an access fault, the level of translation table that gave the fault.
- For a permission fault, including a fault caused by a hierarchical permission, the final level of translation table used for that translation.

SMMU_CB*n*_FSYNR1

The 32-bit SMMU_CB*n*_FSYNR1 register bit assignments are IMPLEMENTATION DEFINED.

17.3.2 SMMU_CBn_IPAFAR, IPA Fault Address Register

The SMMU_CB*n*_IPAFAR characteristics are:

Purpose Records the IPA for a stage 1 followed by stage 2 translation that faults in stage 2. For transactions where the initial translation function is a stage 2 translation context bank, SMMU_CBn_FAR contains the input address, that is, the IPA.

Usage constraints An implementation must implement enough bits to record the maximum IPA size or the maximum upstream address size.

In any implementation that supports only stage 2 translation, or that does not support stage 1 followed by stage 2 translation, this register is a read-only alias of SMMU_CBn_FAR and writes are ignored.

This register is valid only for the following types of stage 2 faults:

- Translation faults.
- Access faults.
- External faults.
- Address size faults.

Additionally, this register is valid for permission faults that occur during stage 1. This applies when all of the following conditions are met:

- Stage 2 SMMU_CBn_FSR.PF==1.
- Stage 2 SMMU_CBn_FSYNR0.NESTED==1.
- Stage 2 SMMU_CBn_FSYNR0.S1PTWF==1.

In all other cases, the value of this register is UNKNOWN.

Configurations The value of *N* depends on the supported translation scheme, as defined by the SMMU_IDR0.PTFS field, and on the translation granule size, as defined by the SMMU_IDR2.PTFSv8 bits, and is determined as follows:

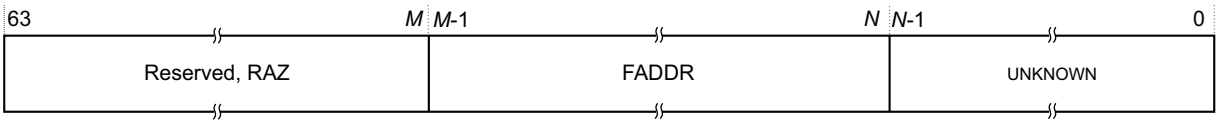
```
if SMMU_IDR0.PTFS == 0b0x or SMMU_IDR2.PTFSv8_4KB == 1 then
    N = 12;
elseif SMMU_IDR2.PTFSv8_16KB == 1 then
    N = 14;
else
    //There must be at least one granule size supported
    assert (SMMU_IDR2.PTFSv8_64KB == 1);
    N = 16;
```

The value of M is determined from the maximum address width encoded by the maximum of SMMU_IDR2.UBS and SMMU_IDR2.IAS,

This register is not provided in SMMUv1.

Attributes A 64-bit register.

The SMMU_CB*n*_IPAFAR bit assignments are:



Bits[63:*M*] Reserved.

FADDR, bits[*M*-1:N]

Fault address, the IPA of the faulting access.

Bits[*N*-1:0] UNKNOWN

17.3.3 SMMU_CB*n*_SCTLR, System Control Register

The SMMU_CB*n*_SCTLR characteristics are:

Purpose Provides top-level control of the translation system for stage 2 translation context bank *n*.

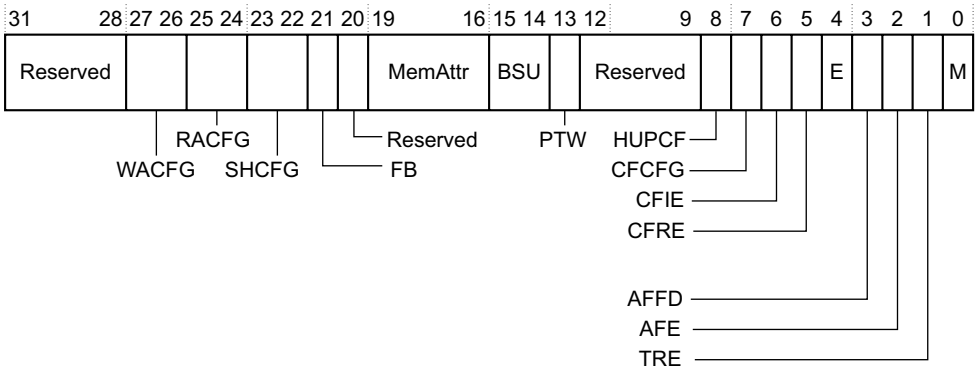
Usage constraints An SMMU implementation might configure certain attributes as RAZ/WI if the downstream bus system does not support this bus marking. See *Memory type and shareability attribute determination on page 2-59* for more information.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 2 translation. See:

- SMMU_IDR0.S2TS.
- SMMU_IDR0.NTS.
- *Translation context bank on page 2-55*.

Attributes A 32-bit RW register.

The SMMU_CB*n*_SCTLR bit assignments are:



Bits [31:28] Reserved.

WACFG, bits[27:26]

Write Allocate Configuration, controls the allocation hint for a write transaction where the translation context bank translation is disabled. That is, where SMMU_CB*n*_SCTLR.M==0.

The encoding of this field is:

`0b00` Default allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Write-Allocate.

`0b11` No Write-Allocate.

In an implementation that does not support Write-Allocate configuration, this field is RAZ/WI.

RACFG, bits[25:24]

Read Allocate Configuration, controls the allocation hint for read transactions where the translation context bank translation is disabled. That is, where SMMU_CB*n*_SCTLR.M==0.

The encoding of this field is:

`0b00` Default allocation attributes.

`0b01` Reserved. See *Reserved memory type and shareability attributes on page 2-62*.

`0b10` Read-Allocate.

`0b11` No Read-Allocate.

In an implementation that does not support Read-Allocate configuration, this field is RAZ/WI.

SHCFG, bits[23:22]

Shared Configuration, controls the shareable attributes of a transaction where the translation context bank is disabled. That is, where SMMU_CB*n*_SCTLR.M==0.

The encoding of this field is:

`0b00` Reserved, in SMMUv1.
Non-shareable, in SMMUv2.

`0b01` Outer Shareable.

`0b10` Inner Shareable.

`0b11` Non-shareable.

This field differs from the equivalent field in the stage 1 format SMMU_CB*n*_SCTLR register. The stage 2 SMMU_CB*n*_SCTLR.SHCFG field is combined with the shared attributes of the previous translation step. See Table 10-2 on page 10-245 for more information.

In an implementation that does not support Shared configuration, this field is RAZ/WI.

FB, bit[21] Force Broadcast, forces the Broadcast of TLB maintenance, BPIALL, and ICIALLU operations.

Bit[20] Reserved.

MemAttr, bits[19:16]

Memory Attributes.

The memory attributes are permitted to be overlaid, if SMMU_CB*n*_SCTLR.M==0.

Memory attribute, MemAttr on page 9-179 describes the MemAttr field encoding.

This field differs from the equivalent field in the stage 1 SMMU_CB*n*_SCTLR format. The stage 2 MemAttr field is combined with the memory attributes presented from the previous translation step. See Table 10-2 on page 10-245.

BSU, bits[15:14]

Barrier Shareability Upgrade, upgrades the required shareability domain of barriers issued by client devices mapped to this Stream mapping register group, by setting the minimum shareability domain that is applied to any barrier.

The encoding of this field is:

`0b00` No effect.

`0b01` Inner Shareable.

`0b10` Outer Shareable.

`0b11` Full system.

The upgrade of the barrier shareability domain might not be supported in all system topologies. In an implementation that does not have this upgrade behavior, this field is RAZ/SBZP.

PTW, bit[13] Protected Translation Walk, only valid for an implementation that has stage 1 followed by stage 2 translation. In an implementation that does not have stage 1 followed by stage 2 translation, it is RES0.

The possible values of this bit are:

0 This behavior is not enabled.

1 Raise a stage 2 permission fault if a stage 1 translation walk is to an area of memory that has the Device or Strongly-ordered memory attribute in the stage 2 translation tables.

Bit[12] Reserved.

The stage 2 translation context bank format does not provide the ASIDPNE field that exists at this location in the stage 1 translation context bank format. For a stage 2 format translation context bank, this field is UNK/SBZP.

Bits[11:9] Reserved.

The stage 2 translation context bank format does not provide the WXN and UWXN fields. For a stage 2 format translation context bank, these fields are UNK/SBZP.

HUPCF, bit[8]

Hit Under Previous Context Fault.

The possible values of this bit are:

0 Stall or terminate any subsequent transaction in the presence of an outstanding context fault.

1 Process any subsequent transaction independently of any outstanding context fault.

The distributed nature of some SMMU implementations means there can be significant delays between detection of a context fault for one upstream client device and transactions from other upstream client devices being stopped.

When this bit is set to 0, it is not guaranteed that faults from different devices are recorded in absolute temporal order. When a device faults, although the setting of this bit is obeyed for that device, no temporal guarantees are made regarding suspension of transactions from different non-faulting devices.

CFCFG, bit[7]

Context Fault Configuration.

The possible values of this bit are:

0 Terminate the transaction.

1 Stall the transaction.

CFIE, bit[6] Context Fault Interrupt Enable.

The possible values of this bit are:

0 Do not raise an interrupt when a context fault occurs.

1 Raise an interrupt when a context fault occurs.

This field resets to 0.

CFRE, bit[5] Context Fault Report Enable.

The possible values of this bit are:

0 Do not return an abort when a Context fault occurs.

1 Return an abort when a Context fault occurs.

E, bit[4] Endianness, indicates the endianness of translation table entries.

The possible values of this bit are:

0 Little endian format.

1 Big endian format.

AFFD, bit[3] Access Flag Fault Disable, determines whether Access flag faults are enabled. Only applicable when AFE==1.

The possible values of this bit are:

0 Access flag faults are enabled.

1 Access flag faults are not enabled.

If enabled, Access flag faults are reported by SMMU_CBn_FSR.

If AFFD==0, AP[0]==0 in the translation table entry causes an Access flag fault, which SMMU_CBn_FSR reports.

If AFFD==1, hardware behaves as if AP[0]==1 regardless of the translation table entry value.

If the SMMU shares translation tables with the processor, any descriptor that is configured with AP[0] as an access flag, that is, when the effective value of AFE is 1, is not held in a processor TLB entry. However, such descriptors might be held in SMMU TLBs. Therefore, any maintenance software that modifies descriptors must be aware that an obsolete descriptor might be cached in the SMMU TLB.

For more information about the *Access permission* (AP) bit, see the *ARM® Architecture Reference Manual, ARMv7-A and ARMv7-R edition*.

AFE, bit[2] Access Flag Enable. This bit is UNK/SBOP.

Stage 2 translation supports only the AArch32 Long-descriptor or the AArch64 translation scheme, meaning that the descriptors support only the simplified model for access permissions. ARM recommends that software treats this bit as UNK/SBOP.

TRE, bit[1] TEX Remap Enable. This bit is UNK/SBOP.

Stage 2 translation supports only the AArch32 Long-descriptor or the AArch64 translation scheme, meaning that the descriptors support only the simplified model for access permissions. ARM recommends that software treats this bit as UNK/SBOP.

M, bit[0] MMU Enable, a global enable bit for the involved translation context bank.

The possible values of this bit are:

0 MMU behavior for this translation context bank is disabled.

1 MMU behavior for this translation context bank is enabled.

This field is not cacheable in TLB entries.

When this bit is changed:

- Any transaction that arrives at the SMMU after the write that updated SMMU_CBn_SCTLR.M completed is guaranteed to use the new value.
- Any transaction that arrived at the SMMU before the completion of the update of SMMU_CBn_SCTLR.M might be processed using either the old value or the new value.

Note

- A TLB invalidate sequence that invalidates all transactions from this bank that might be in progress, and is issued after a completed SMMU_CBn_SCTLR.M write that changes the value from 1 to 0, will complete only after the completion of all non-stalled transactions that were processed using M==1.
- There is no mechanism by which software can check for the completion of stalled transactions that were part-processed using M==1 before they stalled.

17.3.4 SMMU_CBn_TLBIIPAS2, Invalidate TLB by IPA

The SMMU_CB*n*_TLBIIPAS2 characteristics are:

Purpose Invalidates all unlocked TLB entries that match the IPA provided. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This operation must apply to all unlocked entries associated with the specified IPA.

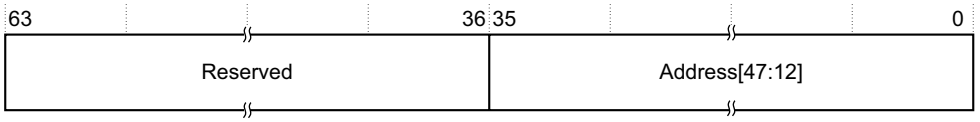
This register must apply to TLB entries containing only stage 2 translation information. It is not required to apply to TLB entries that combine stage 1 and stage 2 if those entries do not store the IPA. For TLB entries that combine stage 1 and stage 2 information and store the IPA it must apply to matching entries that have the VMID specified by the stage 2 translation bank.

An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations This register is not provided in SMMUv1.

Attributes A 64-bit WO register.

The SMMU_CB*n*_TLBIIPAS2 bit assignments are:



Bits[63:36] Reserved.

Address[47:12], bits[35:0]

Bits[47:12] of the address to be invalidated.

- If the translation granule is 64KB, then bits corresponding to Address[15:12] are ignored.
- If the translation granule is 16KB, then bits corresponding to Address[13:12] are ignored.
- For the AArch32 Long-descriptor translation scheme, bits[47:40] are treated as zero.

If an IPA is supplied above the address space size indicated by SMMU_IDR0.IAS, then the extra bits are ignored by the implementation.

To change a stage 2 translation table entry, software must:

1. Perform a stage 2 SMMU_CBn_TLBIIPAS2 operation.
2. Perform a SMMU_CBn_TLBSYNC operation and wait for its completion. For example, write to SMMU_CBn_TLBSYNC and poll SMMU_CBn_TLBSTATUS until it indicates that the synchronization has completed.
3. Invalidate all stage 1 contexts that are mapped to the stage 2 context, using SMMU_TLBIVMIDS1 operation.

17.3.5 SMMU_CBn_TLBIIPAS2L, Invalidate TLB by IPA, Last level

The SMMU_CB*n*_TLBIIPAS2L characteristics are:

Purpose Invalidates any unlocked TLB entries that match the IPA provided and include information from the final level of translation table lookup. See *TLB maintenance operations on page 5-137* for more information.

Usage constraints This register must apply to all unlocked entries associated with the specified IPA.

This register must apply to TLB entries containing only stage 2 translation information. It is not required to apply to TLB entries that combine stage 1 and stage 2 if those entries do not store the IPA. For TLB entries that combine stage 1 and stage 2 information and store the IPA it must apply to matching entries that have the VMID specified by the stage 2 translation bank.

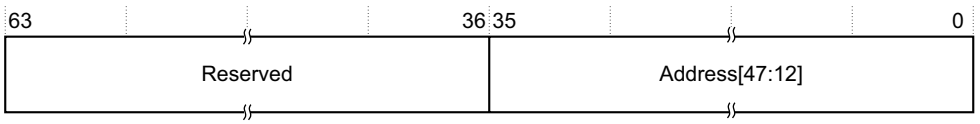
An SMMU implementation can over-invalidate, removing any arbitrary set of unlocked TLB entries, including those allocated from other context banks.

Configurations This register is not provided in SMMUv1.

SMMU_IDR2.IAS specifies the implemented input address space.

Attributes A 64-bit WO register.

The SMMU_CB*n*_TLBIIPAS2L bit assignments are:



Bits[63:36] Reserved.

Address[47:12], bits[35:0]

Bits[47:12] of the address to be invalidated.

- If the translation granule is 64KB, then bits corresponding to Address[15:12] are ignored.
- If the translation granule is 16KB, then bits corresponding to Address[13:12] are ignored.
- For the AArch32 Long-descriptor translation scheme, bits[47:40] are treated as zero.

If an IPA is supplied above the address space size indicated by SMMU_IDR0.IAS, then the extra bits are ignored by the implementation.

To change a stage 2 translation table entry, software must:

1. Perform a stage 2 SMMU_CB*n*_TLBIIPAS2L operation.
2. Perform a SMMU_CBn_TLBSYNC operation and wait for its completion. For example, write to SMMU_CBn_TLBSYNC and poll SMMU_CBn_TLBSTATUS until it indicates that the synchronization has completed.
3. Invalidate all stage 1 contexts that are mapped to the stage 2 context, using SMMU_TLBIVMIDS1 operation.

17.3.6 SMMU_CB*n*_TLBSTATUS, TLB Status

The SMMU_CB*n*_TLBSTATUS characteristics are:

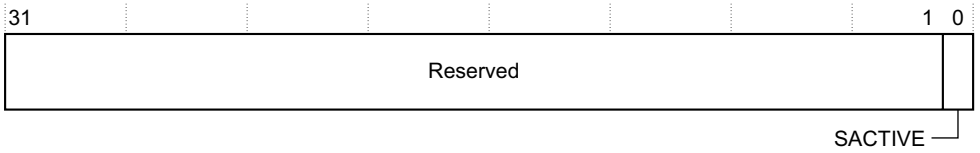
Purpose Indicates the status of any TLB maintenance operations issued before the most recent SMMU_CBn_TLBSYNC operation.

Usage constraints No usage constraints apply.

Configurations This register is not provided in SMMUv1.

Attributes A 32-bit RO register that resets to 0. Writes are ignored. See also Table 17-1 on page 17-345.

The SMMU_CB*n*_STATUS bit assignments are:



Bits[31:1] Reserved.

SACTIVE, bit[0]

SMMU_CBn_TLBSYNC operation active.

The possible values of this bit are:

0 All TLB invalidate operations issued before the most recent SMMU_CBn_TLBSYNC operation have completed.

1 Some TLB invalidate operations issued before the most recent SMMU_CBn_TLBSYNC operation have not completed.

17.3.7 SMMU_CB*n*_TLBSYNC, TLB Synchronize Invalidate

The SMMU_CB*n*_TLBSYNC characteristics are:

Purpose Initiates a synchronization operation that ensures the completion of any TLB invalidate operations previously accepted in the corresponding translation context bank.

Usage constraints This operation operates in the scope of the translation context bank it resides in. After being accepted, the operation does not complete until all translation context bank TLB invalidate operations accepted by the SMMU before the synchronize operation was accepted are complete.

See *TLB maintenance operations on page 5-137* for more details about the usage model.

Configurations This register is not provided in SMMUv1.

Attributes A 32-bit WO register. Reads are SBZ. See also Table 17-1 on page 17-345.

The SMMU_CB*n*_TLBSYNC bit assignments are reserved.

17.3.8 SMMU_CB*n*_TCR, Translation Control Register

The SMMU_CB*n*_TCR characteristics are:

Purpose Provides additional configuration for the stage 2 translation process.

Usage constraints No usage constraints apply.

Configurations The format of this register depends on the value of the SMMU_CBA2Rn.VA64 bit. For more information, see:

- *Register format when the value of SMMU_CBA2Rn.VA64 is 0*.
- *Register format when the value of SMMU_CBA2Rn.VA64 is 1 on page 17-359*.

It is IMPLEMENTATION DEFINED whether the system implements stage 2 translation. See:

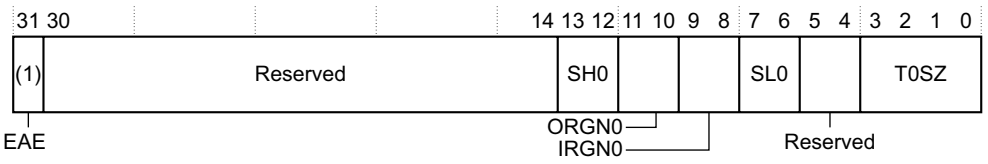
- SMMU_IDR0.S2TS.
- SMMU_IDR0.NTS.
- *Translation context bank on page 2-55*.

In SMMUv1, this register is SMMU_CBn_TTBCR.

Attributes A 32-bit RW register.

Register format when the value of SMMU_CBA2Rn.VA64 is 0

The SMMU_CB*n*_TCR bit assignments in this format are:



EAE(1), bit[31]

Extended Address Enable.

For a stage 2 translation context entry, this field always reads as the value 1. Writes are ignored.

A value of 1 means that the translation system defined in the LPAE is used.

Bits[30:14] Reserved.

SH0, bits[13:12]

Shareability attributes for the memory associated with the translation table walks using SMMU_CBn_TTBR0.

ORGN0, bits[11:10]

Outer cacheability attributes for the memory associated with the translation table walks using SMMU_CBn_TTBR0.

IRGN0, bits[9:8]

Inner cacheability attributes for the memory associated with the translation table walks using SMMU_CBn_TTBR0.

SL0, bits[7:6] Lookup Start Level for the SMMU_CBn_TTBR0 addressed region. The encoding of this field is:

- **0** Level 2.
- **1** Level 1.
- **2** UNPREDICTABLE.
- **3** UNPREDICTABLE.

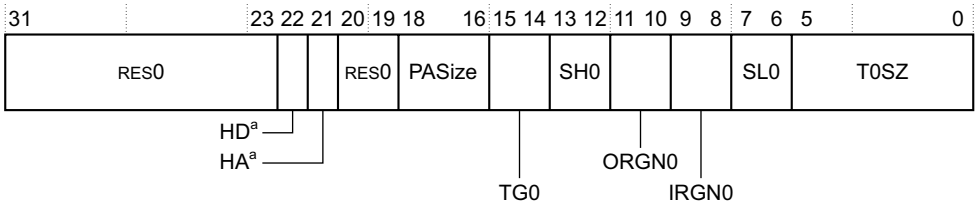
Bit[5:4] Reserved. Writes to bit[4] are ignored.

T0SZ, bits[3:0]

The Size offset of the SMMU_CBn_TTBR0 addressed region, encoded as a 4-bit signed number giving the size of the region as $2^{32-T0SZ}$.

Register format when the value of SMMU_CBA2Rn.VA64 is 1

The SMMU_CB*n*_TCR bit assignments in this format are:



a. Reserved in SMMUv1

Bits[31:23] RES0.

HD, bit[22], SMMUv2 only

Hardware management of Dirty bit. The possible values of this bit are:

0 Stage 2 Dirty bit update disabled.

1 Stage 2 Dirty bit update enabled if the value of the HA bit 1.

If the value of SMMU_IDR1.HAFDBS is 0 or 1, this bit is RES0.

HA, bit[21], SMMUv2 only

Hardware management of Access flag. The possible values of this bit are:

0 Stage 2 Access flag update disabled.

1 Stage 2 Access flag update enabled.

If the value of SMMU_IDR1.HAFDBS is 0, this bit is RES0.

Bits[22:21], SMMUv1

RES0.

Bits[20:19] RES0.

PASize, bits[18:16]

Physical address size. If the SMMU supports a larger address space than a device provides, this field enables software to reduce the amount of SMMU address space.

The encoding of this field is:

`0b000` 32-bits, 4GB

`0b001` 36-bits, 64GB

`0b010` 40-bits, 1TB

`0b011` 42-bits, 4TB

`0b100` 44-bits, 16TB

`0b101` 48-bits, 256TB

All other values are reserved, and treated as 48-bits.

TG0, Bits[15:14]

Translation granule size for SMMU_CBn_TTBR0, for stage 2 translations. The possible values of this field are:

`0b00` 4KB granule size.

`0b01` 64KB granule size.

`0b10` 16KB granule size.

`0b11` Reserved.

If software sets this field to a reserved value, or a size that has not been implemented, hardware treats the field as if it is programmed to an IMPLEMENTATION DEFINED choice of the sizes that are implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

SH0, bits[13:12]

Shareability attributes for the memory associated with the translation table walks using SMMU_CBn_TTBR0.

ORGN0, bits[11:10]

Outer cacheability attributes for the memory associated with the translation table walks using SMMU_CBn_TTBR0.

IRGN0, bits[9:8]

Inner cacheability attributes for the memory associated with the translation table walks using SMMU_CBn_TTBR0.

SL0, bits[7:6] Lookup Start Level for the SMMU_CBn_TTBR0 addressed region. The encoding of this field depends on the translation granule size indicated by the TG0 field:

4KB translation granule

`0b00` Level 2.

`0b01` Level 1.

`0b10` Level 0.

16KB or 64KB translation granule

`0b00` Level 3.

`0b01` Level 2.

`0b10` Level 1.

T0SZ, bits[5:0]

The Size offset of the SMMU_CBn_TTBR0 addressed region, encoded as a 6-bit signed number giving the size of the region as $2^{(64-T0SZ)}$.

17.3.9 SMMU_CB*n*_TTBR0, Translation Table Base Register

The SMMU_CB*n*_TTBR0 characteristics are:

Purpose Holds the base address of translation table 0.

Usage constraints For a stage 2 translation context bank, only the AArch32 Long-descriptor and the AArch64 translation schemes are supported.

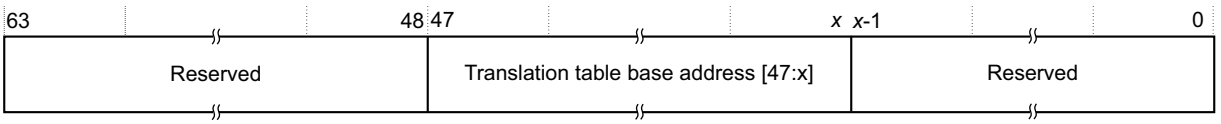
Depending on the translation scheme selected, either an address size fault or a translation fault is generated if a PA exceeds the number of bits that the SMMU_IDR2.OAS field defines.

Configurations It is IMPLEMENTATION DEFINED whether the system implements stage 2 translation. See:

- SMMU_IDR0.S2TS.
- SMMU_IDR0.NTS.
- *Translation context bank on page 2-55*.

Attributes A 64-bit RW register.

The SMMU_CB*n*_TTBR0 bit assignments are:



Bits[63:48] Reserved.

Translation table base address [47:*x*], bits[47:*x*]

Translation table base address, bits[47:*x*].

The value *x* is determined by SMMU_CBn_TCR.T0SZ according to the algorithm specified in *The algorithm for finding the translation table descriptors on page 2-79*.

The number of implemented bits is determined by the maximum address size, as defined by the SMMU_IDR2.IAS and SMMU_IDR2.OAS fields.

For the AArch32 Long-descriptor translation scheme, bits[47:40] are treated as zero.

Bits[*x*-1:0] Reserved.

Appendix A
Register Names

This appendix provides information about differences in register names between SMMU architecture versions. It also provides naming information for any SMMU registers where a similar register exists in the ARM architecture. It contains the following section:

- *Summary of corresponding SMMU and ARM registers on page A-364*.

A.1 Summary of corresponding SMMU and ARM registers

Table A-1 shows the SMMU registers where registers on an ARM processor perform a similar role.

Table A-1 Corresponding SMMU and ARM processor registers

		ARM architecture register	
SMMU register	**Short description**	**AArch64**	**AArch32**
SMMU_CBn_CONTEXTIDR	Context ID	CONTEXTIDR_EL1	CONTEXTIDR
SMMU_CBn_FAR	Fault Address Register	FAR_EL1 FAR_EL2 FAR_EL3 HPFAR_EL2	DFAR, IFAR HDFAR, HIFAR FAR_EL3 HPFAR
SMMU_CBn_MAIRm	Memory Attribute Indirection Registers	MAIR_EL1 MAIR_EL2 MAIR_EL3	HMAIRn
SMMU_CBn_TCR[a] SMMU_CBn_TCR2	Translation Control Register	TCR_EL1 TCR_EL2 TCR_EL3	TTBCR(NS) HTCR TTBCR(S)
SMMU_CBn_TTBR0 SMMU_CBn_TTBR1	Translation Table Base Register	TTBR0_EL1 TTBR0_EL2 TTBR1_EL1	TTBR0 TTBR1
SMMU_CBn_TCR[a] SMMU_CBn_TCR2	EL1&0 stage 2 Translation Control Register	VTCR_EL2	VTCR
SMMU_CBn_TTBR0	EL1&0 stage 2 Translation Table Base Register	VTTBR_EL2	VTTBR

a. In SMMUv1, SMMU_CBn_TTBCR.

Glossary

Abort An exception caused by an illegal memory access. Aborts can be caused by the external memory system or the MMU.

See also Data abort, External abort, and Prefetch abort.

Banked registers
A register that has multiple instances, with the instance that is in use depending on the state of the SMMU or the properties of the access made to the register.

Byte An 8-bit data item.

Data abort An indication from a memory system to the processor of an attempt to access an illegal data memory location.

See also Abort, External abort, and Prefetch abort.

Deprecated Something that is present in the ARM architecture for backwards compatibility. Whenever possible software must avoid using deprecated features. Features that are deprecated but are not optional are present in current implementations of the ARM architecture, but might not be present, or might be deprecated and OPTIONAL, in future versions of the ARM architecture.

See also OPTIONAL.

Exception Handles an event. For example, an exception could handle an external interrupt or an undefined instruction.

External abort An abort generated by the external memory system.

See also Abort and Data abort.

Halfword-aligned
A data item having a memory address that is divisible by 2.

IMP An abbreviation used in diagrams to indicate that one or more bits have IMPLEMENTATION DEFINED behavior.

IMPLEMENTATION DEFINED
Behavior that is not architecturally defined, but must be defined and documented by individual implementations. In body text, the term IMPLEMENTATION DEFINED is shown in SMALL CAPITALS.

IMPLEMENTATION SPECIFIC
Behavior that is not architecturally defined, and might not be documented by an individual implementations. Used when there are a number of implementation options available and the option chosen does not affect software compatibility.

When IMPLEMENTATION SPECIFIC appears in body text, it is always in SMALL CAPITALS.

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) and Virtual address (VA).

IPA *See* Intermediate Physical Address (IPA).

Little-endian memory
Means that:

- A byte or halfword at a word-aligned address is the least significant byte or halfword in the word at that address.
- A byte at a halfword-aligned address is the least significant byte in the halfword at that address.

Memory coherency
The problem of ensuring that when a memory location is read, either by a data read or an instruction fetch, the value actually obtained is always the value that was most recently written to the location. This can be difficult when there are multiple possible physical locations, such a main memory and at least one of a write buffer and one or more levels of cache.

Memory Management Unit (MMU)
Provides detailed control of the part of a memory system that provides a single stage of address translation. Most of the control is provided using translation tables that are held in memory, and define the attributes of different regions of the physical memory map.

MMU *See* Memory Management Unit (MMU).

Offset addressing
Means that the memory address is formed by adding or subtracting an offset to or from the base register value.

OPTIONAL When applied to a feature of the architecture, OPTIONAL indicates a feature that is not required in an implementation of the ARM architecture:

- If a feature is OPTIONAL and deprecated, this indicates that the feature is being phased out of the architecture. ARM expects such a feature to be included in a new implementation only if there is a known backwards-compatibility reason for the inclusion of the feature.

  A feature that is OPTIONAL and deprecated might not be present in future versions of the architecture.
- A feature that is OPTIONAL but not deprecated is, typically, a feature added to a version of the ARM architecture after the initial release of that version of the architecture. ARM recommends that such features are included in all new implementations of the architecture.

In body text, these meanings of the term OPTIONAL are shown in SMALL CAPITALS.

Note: Do not confuse these ARM-specific uses of OPTIONAL with other uses of *optional*, where it has its usual meaning. These include:

- Optional arguments in the syntax of many instructions.
- Behavior determined by an implementation choice, for example the optional byte order reversal in an ARMv7-R implementation, where the SCTLR.IE bit indicates the implemented option.

See also Deprecated.

PA *See* Physical address (PA).

Physical address (PA)
An address that identifies a location in the physical memory map.

See also Intermediate Physical Address (IPA), Virtual address (VA).

Prefetch abort
An indication from the internal or external memory system to the processor that an instruction has been fetched from an illegal memory location. An exception is taken only if the processor attempts to execute the instruction. No exception is taken if the processor does not execute an instruction prefetched from a faulting memory location.

See also Abort and Data abort.

RAO
See Read-As-One (RAO).

RAZ
See Read-As-Zero (RAZ).

RAZ/SBZP
In versions of the ARM architecture before ARMv8, Read-As-Zero, Should-Be-Zero-or-Preserved on writes.

In ARMv8, RES0 replaces this description.

See also UNK/SBZP, Read-As-Zero (RAZ) and Should-Be-Zero-or-Preserved (SBZP).

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-allocate cache
A cache in which a cache miss on reading data causes a cache line to be allocated into the cache.

Read-As-One (RAO)
Hardware must implement the field as reading as all 1s.

Software:

- Can rely on the field reading as all 1s.
- Must use a SBOP policy to write to the field.

This description can apply to a single bit that reads as 1, or to a field that reads as all 1s.

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.

Reserved
Unless otherwise stated in the architecture or product documentation, reserved:
- Instruction and 32-bit system control register encodings are UNPREDICTABLE.
- 64-bit system control register encodings are UNDEFINED.
- Register bit fields are UNK/SBZP.

RES0
Within the architecture, there are some cases where a register bit or bitfield:

- Is RES0 in some defined architectural context.
- Has different defined behavior in a different architectural context.

This means the definition of RES0 is:

If a bit is RES0 in all contexts

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.

   The bit might be described as RES0, WI, to distinguish it from a bit that behaves as described in 2.
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 to the bit.

     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.
   - 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.

Whether the 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

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, 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 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.

For any RES0 bit, software:

- Must not rely on the bit reading as 0.
- Must use an SBZP policy to write to the bit.

The RES0 description can apply to bits or bitfields 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.

This RES0 description can apply to a single bit that should be written as its preserved value or as 0, or to a field that should be written as its preserved value or as all 0s.

In body text, the term RES0 is shown in SMALL CAPITALS.

See also Read-As-Zero (RAZ),Should-Be-Zero-or-Preserved (SBZP),UNKNOWN.

SBO *See* Should-Be-One (SBO).

SBOP *See* Should-Be-One-or-Preserved (SBOP).

SBZ *See* Should-Be-Zero (SBZ).

SBZP *See* Should-Be-Zero-or-Preserved (SBZP).

Security hole A mechanism by which execution at the current level of privilege can achieve an outcome that cannot be achieved at the current or lower level of privilege using instructions that are not UNPREDICTABLE. The ARM architecture forbids security holes.

Security State Determination (SSD)
Provides a means of identifying whether an upstream device or stream is Secure or Non-secure.

Should-Be-One (SBO)
Hardware must ignore writes to the field.

Software should write the field as all 1s. If software writes a value that is not all 1s, it must expect an UNPREDICTABLE result.

This description can apply to a single bit that should be written as 1, or to a field that should be written as all 1s.

Should-Be-One-or-Preserved (SBOP)
From the introduction of the ARMv7 Large Physical Address Extension the definition of SBOP is modified for register bits that are SBOP in some but not all contexts. The generic definition of SBOP given here applies only to bits that are not affected by this modification.

Hardware must ignore writes to the field.

If software has read the field since the PE implementing the field was last reset and initialized, it must preserve the value of the field by writing the value that it previously read from the field. Otherwise, it must write the field as all 1s.

If software writes a value to the field that is not a value previously read for the field and is not all 1s, it must expect an UNPREDICTABLE result.

This description can apply to a single bit that should be written as its preserved value or as 1, or to a field that should be written as its preserved value or as all 1s.

See also Should-Be-Zero-or-Preserved (SBZP) and Should-Be-One (SBO).

Should-Be-Zero (SBZ)
Hardware must ignore writes to the field.

ARM strongly recommends that software writes the field as all 0s. If software writes a value that is not all 0s, it must expect an UNPREDICTABLE result.

This description can apply to a single bit that should be written as 0, or to a field that should be written as all 0s.

Should-Be-Zero-or-Preserved (SBZP)
From the introduction of ARMv7 Large Physical Address Extension, the definition of SBZP is modified for register bits that are SBZP in some but not all contexts. The generic definition of SBZP given here applies only to bits that are not affected by this modification.

Hardware must ignore writes to the field.

If software has read the field since the PE implementing the field was last reset and initialized, it must preserve the value of the field by writing the value that it previously read from the field. Otherwise, it must write the field as all 0s.

If software writes a value to the field that is not a value previously read for the field and is not all 0s, it must expect an UNPREDICTABLE result.

This description can apply to a single bit that should be written as its preserved value or as 0, or to a field that should be written as its preserved value or as all 0s.

See also Should-Be-One-or-Preserved (SBOP) and Should-Be-Zero (SBZ).

SSD *See* Security State Determination (SSD).

SSD security state
Indicates whether a particular device or stream is in the Secure or the Non-secure domain at the time it initiates a memory access. For a given transaction, the upstream device is either SSD Secure or SSD Non-secure.

TLB *See* Translation Lookaside Buffer (TLB).

Transaction security state
Whether a transaction originates from a Secure or a Non-secure device. A transaction from a Secure device can make both Secure and Non-secure MMU accesses. A transaction from a Non-secure device can only make Non-secure accesses.

Translation Lookaside Buffer (TLB)
A memory structure containing the results of translation table walks. They help to reduce the average cost of memory accesses. Usually, there is a TLB for each memory interface of the ARM implementation.

Translation table A table held in memory that defines the properties of memory areas of various sizes from 1KB to 1MB.

Translation table walk
The process of doing a full translation table lookup. It is performed automatically by hardware.

UNDEFINED Indicates an instruction that generates an Undefined Instruction exception.

In body text, the term UNDEFINED is shown in SMALL CAPITALS.

UNK/SBOP Hardware must implement the field as Read-As-One, and must ignore writes to the field.

Software must not rely on the field reading as all 1s, and except for writing back to the register it must treat the value as if it is UNKNOWN. Software must use an SBOP policy to write to the field.

This description can apply to a single bit that should be written as its preserved value or as 1, or to a field that should be written as its preserved value or as all 1s.

See also Read-As-One (RAO), Should-Be-One-or-Preserved (SBOP), and UNKNOWN.

UNK/SBZP Hardware must implement the field as Read-As-Zero, and must ignore writes to the field.

Software must not rely on the field reading as all 0s, and except for writing back to the register it must treat the value as if it is UNKNOWN. Software must use an SBZP policy to write to the field.

This description can apply to a single bit that should be written as its preserved value or as 0, or to a field that should be written as its preserved value or as all 0s.

See also Read-As-Zero (RAZ), Should-Be-Zero-or-Preserved (SBZP), and UNKNOWN.

UNK An abbreviation indicating that software must treat a field as containing an UNKNOWN value.

In any implementation, the bit must read as 0, or all 0s for a bit field. Software must not rely on the field reading as zero.

See also UNKNOWN.

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 be a security hole.

An UNKNOWN value must not be documented or promoted as having a defined value or effect.

When UNKNOWN appears in body text, it is always in SMALL CAPITALS.

UNPREDICTABLE
Means the behavior cannot be relied on. 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 in a Non-secure EL1 or EL0 mode of an instruction that is UNPREDICTABLE can be implemented as generating a Hyp Trap exception, provided that at least one instruction that is not UNPREDICTABLE causes a Hyp Trap exception.

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).

Word A 32-bit data item. Words are normally word-aligned in ARM systems.

Word-aligned A data item having a memory address that is divisible by four.

Write-Allocate cache
A cache in which a cache miss on storing data causes a cache line to be allocated into the cache.

Write-Back cache
A cache in which when a cache hit occurs on a store access, the data is only written to the cache. Data in the cache can therefore be more up-to-date than data in main memory. Any such data is written back to main memory when the cache line is cleaned or re-allocated. Another common term for a Write-Back cache is a *copy-back cache*.

Write-Through cache
A cache in which when a cache hit occurs on a store access, the data is written both to the cache and to main memory. This is normally done using a write buffer, to avoid slowing down the PE.