ARM® System Memory Management Unit Architecture Specification

SMMU architecture version 3.0 and version 3.1

Copyright © 2016-2017 ARM Limited or its affiliates. All rights reserved.
Document number: IHI 0070B

System Memory Management Unit Architecture Specification

Copyright © 2016-2017 ARM Limited or its affiliates. All rights reserved.

Release information

Changes made to this document:

Date	Issue	Confidentiality	Change
15 September 2016	A	Non-Confidential	First release
14 June 2017	B	Non-Confidential	Amendments and clarifications.

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM Limited ("ARM**"). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, ARM makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version shall prevail.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to ARM's customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement specifically covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the

trademarks of their respective owners. You must follow the ARM trademark usage guidelines http://www.arm.com/about/trademarks/guidelines/index.php.

Copyright © 2016-2017 ARM Limited or its affiliates. All rights reserved.

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20327

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

Confidentiality Status

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

Product Status

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

Web Address

http://www.arm.com

Contents

CONTENTS 4

1 ABOUT THIS DOCUMENT 14

1.1 References 14

1.2 Terms and abbreviations 14

1.3 Document Scope 17

2 INTRODUCTION 18

2.1 History 18

2.2 SMMUv3.0 features 19

2.3 SMMUv3.1 features 20

2.4 System placement 21

3 OPERATION 24

3.1 Software interface 24

3.2 Stream numbering 24

3.3 Data structures and translation procedure 25
3.3.1 Stream Table lookup 26
3.3.2 StreamIDs to Context Descriptors 28
3.3.3 Configuration and Translation lookup 32
3.3.4 Transaction attributes: incoming, two-stage translation and overrides 35
3.3.5 Translation table descriptors 36

3.4 Address sizes 36
3.4.1 Input address size and Virtual Address size 38
3.4.2 Address alignment checks 39
3.4.3 Address sizes of SMMU-originated accesses 39

3.5 Command and Event queues 42
3.5.1 SMMU circular queues 42
3.5.2 Queue entry visibility semantics 45
3.5.3 Event queue behavior 45
3.5.4 Definition of event record write “Commit” 46
3.5.5 Event merging 46

3.6 Structure and queue ownership 47

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.7 Programming registers 48

3.8 Virtualization 48

3.9 Support for PCI Express, PASIDs, PRI and ATS 48
3.9.1 ATS Interface 49
3.9.2 Changing ATS configuration 56

3.10 Support for two Security states 57
3.10.1 Secure State Determination (SSD) 58
3.10.2 Secure commands, events and configuration 59

3.11 Reset, Enable and initialization 61

3.12 Fault models, recording and reporting 63
3.12.1 Terminate model 66
3.12.2 Stall model 67
3.12.3 Considerations for client devices using the Stall fault model 70
3.12.4 Virtual Memory paging with SMMU 71
3.12.5 Combinations of fault configuration with two stages 72

3.13 Translation table entries and Access/Dirty flags 73
3.13.1 Software update of flags 74
3.13.2 Access flag hardware update 75
3.13.3 Dirty flag hardware update 75
3.13.4 HTTU behavior summary 76
3.13.5 HTTU with two stages of translation 76
3.13.6 ATS, PRI and translation table flag update 77
3.13.7 Hardware flag update for Cache Maintenance Operations and Destructive Reads 79

3.14 Speculative accesses 79

3.15 Coherency considerations and memory access types 80
3.15.1 Client devices 81

3.16 Embedded implementations 81
3.16.1 Changes to structure and queue storage behavior when fixed/preset 82

3.17 TLB tagging, VMIDs, ASIDs and participation in broadcast TLB maintenance 83
3.17.1 The Global flag in the Translation Table Descriptor 86
3.17.2 TLB maintenance from ARMv8-A PEs with EL3 in AArch64 86
3.17.3 TLB maintenance from ARMv7-A PEs or ARMv8-A PEs with EL3 using AArch32 state 87
3.17.4 Broadcast TLB maintenance in mixed AArch32 and AArch64 systems and with mixed ASID or VMID sizes 87
3.17.5 EL2 ASIDs in EL2 Host (E2H) mode 88
3.17.6 VMID Wildcards 89

3.18 Interrupts and notifications 90
3.18.1 MSI synchronization 91
3.18.2 Interrupt sources 92

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.19 Power control **92**
3.19.1 Dormant state 93

3.20 TLB and configuration cache conflict **94**
3.20.1 TLB conflict 94
3.20.2 Configuration cache conflicts 94

3.21 Structure access rules and update procedures **95**
3.21.1 Translation tables and TLB invalidation completion behavior 95
3.21.2 Queues 96
3.21.3 Configuration structures and configuration invalidation completion 97

3.22 Destructive reads and directed cache prefetch transactions **101**
3.22.1 SMMUv3.1 control of transaction downgrade 103
3.22.2 SMMUv3.1 permissions model 103
3.22.3 SMMUv3.1 memory types and Shareability 105

4 COMMANDS **106**
4.1.1 Command opcodes 106
4.1.2 Submitting commands to the Command queue 107
4.1.3 Command errors 108
4.1.4 Consumption of commands from the Command queue 108
4.1.5 Reserved fields 109
4.1.6 Common fields 109
4.1.7 Out-of-range parameters 109

4.2 Prefetch **110**
4.2.1 CMD_PREFETCH_CONFIG(StreamID, SSec, SubstreamID, SSV) 111
4.2.2 CMD_PREFETCH_ADDR(StreamID, SSec, SubstreamID, SSV, Addr, Size, Stride) 111

4.3 Configuration structure invalidation **112**
4.3.1 CMD_CFGI_STE(StreamID, SSec, Leaf) 113
4.3.2 CMD_CFGI_STE_RANGE(StreamID, SSec, Range) 113
4.3.3 CMD_CFGI_CD(StreamID, SSec, SubstreamID, Leaf) 114
4.3.4 CMD_CFGI_CD_ALL(StreamID, SSec) 115
4.3.5 CMD_CFGI_ALL(SSec) 116
4.3.6 Action of VM guest OS structure invalidations by hypervisor 117
4.3.7 Configuration structure invalidation semantics/rules 117

4.4 TLB invalidation **118**
4.4.1 TLB invalidation of stage 1 118
4.4.2 TLB invalidation of stage 2 123
4.4.3 Common TLB invalidation 124

4.5 ATS and PRI **124**
4.5.1 CMD_ATC_INV(StreamID, SubstreamID, SSV, Global, Address, Size) 125
4.5.2 CMD_PRI_RESP(StreamID, SubstreamID, SSV, GroupID, Resp) 127

4.6 Fault response and synchronization commands **128**

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.6.1 CMD_RESUME(StreamID, SSec, STAG, Action, Abort) 128
4.6.2 CMD_STALL_TERM(StreamID, SSec) 130
4.6.3 CMD_SYNC(ComplSignal, MSIAddress, MSIData, MSIWriteAttributes) 133

4.7 Command Consumption summary 137

5 DATA STRUCTURE FORMATS 138

5.1 Level 1 Stream Table Descriptor 139

5.2 Stream Table Entry 140
5.2.1 Validity of STE 166

5.3 Level 1 Context Descriptor 168

5.4 Context Descriptor 169
5.4.1 CD notes 184
5.4.2 Validity of CD 188

5.5 Fault configuration (A,R,S bits) 189

6 MEMORY MAP AND REGISTERS 194

6.1 Memory map 194

6.2 Register overview 194

6.3 Register formats 200
6.3.1 SMMU_IDR0 200
6.3.2 SMMU_IDR1 204
6.3.3 SMMU_IDR2 205
6.3.4 SMMU_IDR3 205
6.3.5 SMMU_IDR4 206
6.3.6 SMMU_IDR5 207
6.3.7 SMMU_IIDR 208
6.3.8 SMMU_AIDR 209
6.3.9 SMMU_CR0 209
6.3.10 SMMU_CR0ACK 214
6.3.11 SMMU_CR1 214
6.3.12 SMMU_CR2 216
6.3.13 SMMU_GBPA 218
6.3.14 SMMU_AGBPA 221
6.3.15 SMMU_STATUSR 221
6.3.16 SMMU_IRQ_CTRL 221
6.3.17 SMMU_IRQ_CTRLACK 222
6.3.18 SMMU_GERROR 222
6.3.19 SMMU_GERRORN 224
6.3.20 SMMU_GERROR_IRQ_CFG0 224
6.3.21 SMMU_GERROR_IRQ_CFG1 225

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6.3.22 SMMU_GERROR_IRQ_CFG2 225
6.3.23 SMMU_STRTAB_BASE 226
6.3.24 SMMU_STRTAB_BASE_CFG 227
6.3.25 SMMU_CMDQ_BASE 228
6.3.26 SMMU_CMDQ_CONS 229
6.3.27 SMMU_CMDQ_PROD 231
6.3.28 SMMU_EVENTQ_BASE 231
6.3.29 SMMU_EVENTQ_CONS 232
6.3.30 SMMU_EVENTQ_PROD 233
6.3.31 SMMU_EVENTQ_IRQ_CFG0, SMMU_EVENTQ_IRQ_CFG1, SMMU_EVENTQ_IRQ_CFG2 234
6.3.32 SMMU_PRIQ_BASE 234
6.3.33 SMMU_PRIQ_CONS 235
6.3.34 SMMU_PRIQ_PROD 236
6.3.35 SMMU_PRIQ_IRQ_CFG0, SMMU_PRIQ_IRQ_CFG1 237
6.3.36 SMMU_PRIQ_IRQ_CFG2 237
6.3.37 SMMU_GATOS_CTRL 237
6.3.38 SMMU_GATOS_SID 239
6.3.39 SMMU_GATOS_ADDR 239
6.3.40 SMMU_GATOS_PAR 240
6.3.41 SMMU_VATOS_SEL 242
6.3.42 SMMU_S_IDR0 242
6.3.43 SMMU_S_IDR1 243
6.3.44 SMMU_S_IDR2 243
6.3.45 SMMU_S_IDR3 243
6.3.46 SMMU_S_IDR4 244
6.3.47 SMMU_S_CR0 244
6.3.48 SMMU_S_CR0ACK 246
6.3.49 SMMU_S_CR1 246
6.3.50 SMMU_S_CR2 248
6.3.51 SMMU_S_GBPA 248
6.3.52 SMMU_S_AGBPA 250
6.3.53 SMMU_S_INIT 250
6.3.54 SMMU_S_IRQ_CTRL 251
6.3.55 SMMU_S_IRQ_CTRLACK 252
6.3.56 SMMU_S_GERROR 252
6.3.57 SMMU_S_GERRORN 253
6.3.58 SMMU_S_GERROR_IRQ_CFG0, SMMU_S_GERROR_IRQ_CFG1, SMMU_S_GERROR_IRQ_CFG2 253
6.3.59 SMMU_S_STRTAB_BASE 253
6.3.60 SMMU_S_STRTAB_BASE_CFG 254
6.3.61 SMMU_S_CMDQ_BASE 256
6.3.62 SMMU_S_CMDQ_CONS 257
6.3.63 SMMU_S_CMDQ_PROD 258
6.3.64 SMMU_S_EVENTQ_BASE 259
6.3.65 SMMU_S_EVENTQ_CONS 260
6.3.66 SMMU_S_EVENTQ_PROD 261
6.3.67 SMMU_S_EVENTQ_IRQ_CFG0, SMMU_S_EVENTQ_IRQ_CFG1, SMMU_S_EVENTQ_IRQ_CFG2261
6.3.68 SMMU_S_GATOS_CTRL 262
6.3.69 SMMU_S_GATOS_SID 263

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6.3.70 SMMU_S_GATOS_ADDR 263
6.3.71 SMMU_S_GATOS_PAR 264
6.3.72 ID_REGS 266
6.3.73 SMMU_VATOS_CTRL, SMMU_VATOS_SID, SMMU_VATOS_ADDR, SMMU_VATOS_PAR 267

7 EVENT QUEUE, FAULTS AND ERRORS 268

7.1 Command queue errors 268

7.2 Event queue recorded faults and events 269
7.2.1 Recording of events and conditions for writing to the Event queue 270
7.2.2 Event queue access external abort 271
7.2.3 Secure and Non-secure Event queues 272

7.3 Event records 272
7.3.1 Event record merging 273
7.3.2 F_UUT 274
7.3.3 C_BAD_STREAMID 275
7.3.4 F_STE_FETCH 275
7.3.5 C_BAD_STE 276
7.3.6 F_BAD_ATS_TREQ 277
7.3.7 F_STREAM_DISABLED 278
7.3.8 F_TRANSL_FORBIDDEN 278
7.3.9 C_BAD_SUBSTREAMID 279
7.3.10 F_CD_FETCH 280
7.3.11 C_BAD_CD 280
7.3.12 F_WALK_EABT 281
7.3.13 F_TRANSLATION 283
7.3.14 F_ADDR_SIZE 284
7.3.15 F_ACCESS 285
7.3.16 F_PERMISSION 285
7.3.17 F_TLB_CONFLICT 286
7.3.18 F_CFG_CONFLICT 287
7.3.19 E_PAGE_REQUEST 288
7.3.20 IMPDEF_EVENTn 288
7.3.21 Event queue record priorities 289

7.4 Event queue overflow 291

7.5 Global error recording 292
7.5.1 GERROR interrupt notification 293

8 PAGE REQUEST QUEUE 294

8.1 PRI queue overflow 296
8.1.1 Recovery procedure 297

8.2 Miscellaneous 297

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

8.3 PRG Response Message codes 299

9 ADDRESS TRANSLATION OPERATIONS 300

9.1 Register usage 303
9.1.1 ATOS_CTRL 303
9.1.2 ATOS_SID 303
9.1.3 ATOS_ADDR 303
9.1.4 ATOS_PAR 305
9.1.5 ATOS_PAR.FAULTCODE encodings 307
9.1.6 SMMU_VATOS_SEL 309

10 PERFORMANCE MONITOR EXTENSION 310

10.1 Support and discovery 310

10.2 Overview of counters and groups 310
10.2.1 Overflow, interrupts and capture 312

10.3 Monitor events 313

10.4 StreamIDs and filtering 315
10.4.1 Counter Group StreamID size 316

10.5 Registers 317
10.5.1 SMMU_PMCGn address map 317
10.5.2 Register details 319

10.6 Support for two Security states 331

11 DEBUG/TRACE 333

12 RELIABILITY, AVAILABILITY AND SERVICEABILITY (RAS) 334

12.1 Error propagation, consumption and containment in the SMMU 334

12.2 Error consumption visible through the SMMU programming interface 335

12.3 Service Failure Mode (SFM) 335

12.4 RAS fault handling/reporting 336

13 ATTRIBUTE TRANSFORMATION 337

13.1 SMMU handling of attributes 338
13.1.1 Attribute definitions 338
13.1.2 Attribute support 339
13.1.3 Default input attributes 341

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

13.1.4 Replace 341
13.1.5 Combine 342
13.1.6 Ensuring consistent output attributes 343

13.2 SMMU disabled global bypass attributes 344

13.3 Translation flow, STE bypasses stage 1 and stage 2 346

13.4 Normal translation flow 347
13.4.1 Stage 1 page permissions 348
13.4.2 Stage 1 memory attributes 348
13.4.3 Stage 2 348
13.4.4 Output 349

13.5 Summary of attribute/permission configuration fields 353

13.6 PCIe and ATS attribute/permissions handling 354
13.6.1 PCIe memory type attributes 354
13.6.2 ATS attribute overview 355
13.6.3 Split-stage (STE.EATS == 0b10) ATS behavior and responses 356
13.6.4 Full ATS skipping stage 1 357
13.6.5 Split-stage ATS skipping stage 1 358

13.7 PCIe permission attribute interpretation 359
13.7.1 Permission attributes granted in ATS Translation Completions 361

14 EXTERNAL INTERFACES 364

14.1 Data path ingress/egress ports 364

14.2 ATS Interface, packets, protocol 364

14.3 SMMU-originated transactions 364

15 TRANSLATION PROCEDURE 365

15.1 Translation procedure charts 365

15.2 Notes on translation procedure charts 372

16 SYSTEM AND IMPLEMENTATION CONSIDERATIONS 373

16.1 Stages 373

16.2 Caching 373
16.2.1 Caching combined structures 374
16.2.2 Data dependencies between structures 374

16.3 Programming implications of bus address sizing 375

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

16.4 System integration 376

16.5 System software 376

16.6 Implementation-defined features 377
16.6.1 Configuration and translation cache locking 377

16.7 Interconnect-specific features 377
16.7.1 Reporting of Unsupported Client Transactions 377
16.7.2 Non-data transfer transactions 378
16.7.3 Treatment of AMBA Exclusives from client devices 380
16.7.4 Treatment of downstream aborts 381
16.7.5 SMMU and AMBA attribute differences 381
16.7.6 Far Atomic operations 384
16.7.7 AMBA DVM messages with respect to CD.ASET == 1 TLB entries 384

17 APPENDIX - THE ARM RAS ARCHITECTURE 386

17.1 Faults, errors, and failures 386

17.2 Error handling and recovery 387

17.3 Fault handling 388

17.4 Nodes 389
17.4.1 Synchronization and error record accesses 390

17.5 Standard error record 391
17.5.1 Security and virtualization 391
17.5.2 Multiple records per node 392
17.5.3 Error types in the error record 392
17.5.4 Writing the error record 393
17.5.5 Prioritizing errors 394
17.5.6 Overwriting the error syndrome 395
17.5.7 Keeping the previous error syndrome 396
17.5.8 Detecting multiple errors 396
17.5.9 Standard format Corrected error counter 396

17.6 Error recovery interrupt 397

17.7 Fault handling interrupt 397

17.8 In-band error signaling 398

17.9 Error recovery and fault handling signaling 398

17.10 Error recovery reset 400

17.11 Memory-mapped view of an error record 401
17.11.1 Supported access sizes 401

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

17.12 Reset values 402
17.12.1 Writes to ERR<n>STATUS 402

17.13 Register index 403

18 APPENDIX - THE ARM RAS REGISTERS 405
18.1.1 ERR<*n*>ADDR, Error Record Address Register 405
18.1.2 ERR<*n*>CTLR, Error Record Control Register 406
18.1.3 ERR<*n*>FR, Error Record Feature Register 410
18.1.4 ERR<*n*>MISC0, Error Record Miscellaneous Register 0 413
18.1.5 ERR<*n*>MISC1, Error Record Miscellaneous Register 1 417
18.1.6 ERR<*n*>STATUS, Error Record Primary Status Register 418
18.1.7 ERRDEVARCH, Device Architecture Register 423
18.1.8 ERRGSR<*n*>, Error Group Status Register 425
18.1.9 ERRIDR, Error Record ID Register 426
18.1.10 ERRIRQCR<*n*>, Error Interrupt Configuration Register 426

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

1 About this document

1.1 References

This document refers to the following documents:

[1] PCI-SIG, “PCI Express Base Specification Revision 3.0”.

[2] PCI-SIG, “Process Address Space ID (PASID)”.

[3] PCI-SIG, “Address Translation Services Revision 1.1”.

[4] ARM, “DDI0487: ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile”.

[5] ARM, “DDI0557: ARM® Architecture Reference Manual, ARMv8.1, for ARMv8-A architecture profile”.

[6] ARM, “IHI0062: ARM® System Memory Management, SMMU architecture version 2.0”.

[7] ARM, “IHI0069: ARM® Generic Interrupt Controller, GIC architecture version 3.0 and version 4.0”.

[8] ARM, “IHI0029: ARM® CoreSight™ Architecture Specification 3.0”.

[9] PCI-SIG, “PASID Translation”.

[10] ARM, “ARM-DEN-0029: Server Base System Architecture”.

[11] ARM, “IHI 0022: AMBA® AXI™ and ACE™ Protocol Specification AXI3™, AXI4™, and AXI4-Lite™, ACE™ and ACE Lite™”.

1.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
ASID	Address Space ID, distinguishing TLB entries for separate address spaces. For example, address spaces of PE processes are distinguished by ASID.
VMID	Virtual Machine ID, distinguishing TLB entries for addresses from separate virtual machines
DVM	Distributed Virtual Memory, a protocol for interconnect messages to provide broadcast TLB maintenance operations (among other things).
VM	Virtual Machine. In this document, VM never means Virtual Memory except when used as part of an existing acronym.
VA	Virtual Address
IPA	Intermediate Physical Address
PA	Physical Address
RC	PCI Express Root Complex [1]

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Endpoint (EP)	A PCI Express [1] function, used in the context of a device that is a client of the SMMU.
PASID	PCI Express term [2], a Process Address Space ID. **Note:** a PASID is an endpoint-local ID so there might be many distinct uses of a specific PASID value in a system. Despite the similarity in name, a PCIe PASID is not the same as a PE ASID, which is intended to be unique within the realm of an Operating System.
ATS	PCI Express term for Address Translation Services [3] provided for remote endpoint TLBs
Split-stage ATS	SMMU facility used with two-stage translation, providing a way to use ATS with stage 1 and use non-ATS translation for stage 2.
PRI	PCI Express term for Page Request Interface, an extension to ATS allowing an endpoint to request an OS to make a paged virtual memory mapping present for DMA.
SMMU	System MMU. Unless otherwise specified, this term is used to mean SMMUv3. Any reference to prior versions of the SMMU specifications is explicitly suffixed with the architecture version number, for example SMMUv1.
ATOS	SMMU facility providing VA-to-IPA/PA translations using system-accessible registers. In addition, VATOS provides a second set of registers for direct use from a virtual machine, with the added constraint that only VA-to-IPA translations can be returned.
PTE	Page Table Entry (typically implies the final or leaf entry of a walk).
Processing Element (PE)	The abstract machine defined in the ARM architecture, as documented in an ARM Architecture Reference Manual [4]. A PE implementation compliant with the ARM architecture must conform with the behaviors described in the corresponding ARM Architecture Reference Manual.
TT	Translation table, synonymous with Page Table, as used by ARM architecture.
TTD	Translation table descriptor, synonymous with Page Table Entry, as used by the ARM architecture
HTTU	Hardware Translation Table Update. The act of updating the Access flag or Dirty state of a page in a given TTD which is automatically done in hardware, on an access or write to the corresponding page.
IMPLEMENTATION DEFINED	Means that the behavior is not architecturally defined, but must be defined and documented by individual implementations. For more information, see [4]. In body text, the term IMPLEMENTATION DEFINED is shown in SMALL CAPITALS.
Implementation specific	Behavior that is not defined by the SMMU architecture, and might not be documented by individual implementations. Used where one of a number of implementation options might be chosen and the option chosen does not affect software compatibility. Software cannot rely on any implementation-specific behavior.
IGNORED	Indicates that the architecture guarantees that the bit or field is not interpreted or modified by hardware. In body text, the term IGNORED is shown in SMALL CAPITALS.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

LPAE	Large Physical Address Extension: The ARMv7 'Long' translation table format, supporting 40-bit output addresses (and 40-bit IPAs) and having 64-bit TTDs – identical to the ARMv8 AArch32 translation table format.
TTW	Translation Table Walk. This is the act of performing a translation by traversing the in-memory tables.
SSD	Secure State Determination. The concept of associating a stream of data from a client device with being under Secure or Non-secure control, and determining a specific Secure or Non-secure configuration for the stream.
STE	Stream table entry.
L1STD	Level-1 Stream Table Descriptor. Used in a 2-level Stream Table.
CD	Context descriptor.
L1CD	Level-1 Context descriptor. Used in a 2-level CD table.
Client device	A device whose incoming traffic to the system is controlled by an SMMU.
Stage 1 Stage 2	One of the two steps of nested translation whereby the output of one set of translation tables is fed into a second set of translation tables. In sequence, stage 1 is the first table indexed, stage 2 is the second.
Bypass	A configuration that passes through a stage of translation without any addresses transformation is using bypass. If an SMMU does not implement a translation stage, that stage is considered equivalent to a bypass configuration.
Stage N-only	A translation configuration for a stream of data in which one of two translation stages is configured to translate and the other is in bypass (whether by configuration or fixed by SMMU implementation).
E2H	EL2 Host Mode. The Virtualization Host Extensions in ARMv8.1 [5] extend the EL2 translation regime providing ASID-tagged translations. In this document, EL2-E2H mode is the abbreviation that is used.
TR	Translation Request, used in the context of a PCIe ATS request to the SMMU, or another distributed implementation making translation requests to a central unit.
UNPREDICTABLE	Means the behavior cannot be relied upon. UNPREDICTABLE behavior must not perform any function that cannot be performed at the current or a lower level of privilege using instructions that are not UNPREDICTABLE. UNPREDICTABLE behavior must not be documented or promoted as having a defined effect. In body text, the term UNPREDICTABLE is shown in SMALL CAPITALS.
CONSTRAINED UNPREDICTABLE	Where an instruction can result in UNPREDICTABLE behavior, the architecture specifies a narrow range of permitted behaviors. This range is the range of CONSTRAINED UNPREDICTABLE behavior. All implementations that are compliant with the architecture must follow the CONSTRAINED UNPREDICTABLE behavior. In body text, the term CONSTRAINED UNPREDICTABLE is shown in SMALL CAPITALS.
UNKNOWN	An UNKNOWN value does not contain valid data, and can vary from moment to moment and implementation to implementation. An UNKNOWN value must not

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

	return information that cannot be accessed at the current or a lower level of privilege of operating software using accesses that are not UNPREDICTABLE and do not return UNKNOWN values. An UNKNOWN value must not be documented or promoted as having a defined value or effect. In body text, the term UNKNOWN is shown in SMALL CAPITALS.
Reserved	Unless otherwise specified, a Reserved field behaves as RES0. For an identification, or otherwise read-only register field, a reserved encoding is never given by the SMMU. For a field that is provided to the SMMU, reserved values must not be used and their behavior must not be relied upon.
RES0	A reserved bit or field with Should-Be-Zero-or-Preserved (SBZP) behavior, or equivalent read-only or write-only behavior. Used for fields in register descriptions, and for fields in architecturally-defined data structures that are held in memory, for example in translation table descriptors. For a full description see [4].
RES1	A reserved bit or field with Should-Be-One-or-Preserved (SBZP) behavior. Used for fields in register descriptions, and for fields in architecturally-defined data structures that are held in memory, for example in translation table descriptors. For a full description see [4].
Terminate	To complete a transaction with a negative status/abort response; the exact details depend on an implementation's interconnect behavior. When a client transaction is said to have been "terminated" by the SMMU, it has been prevented from progressing into the system and an abort response has been issued to the client (if appropriate for the interconnect in use).
ILLEGAL	A set of conditions that make an STE or CD structure illegal. These conditions differ for the individual CDs and STEs, and are described in detail in the relevant CD and STE descriptions. A field in a structure can make the structure ILLEGAL, for example when it contains an incorrect value, only if the field was not IGNORED for other reasons. Attempts to use an ILLEGAL structure generate an error that is specific to the type of structure.

1.3 Document Scope

This document is the specification for a System Memory Management Unit version 3 following on from the previous SMMUv2 architecture [6].

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

2 Introduction

A System Memory Management Unit (SMMU) performs a task that is analogous to that of an MMU in a PE, translating addresses for DMA requests from system I/O devices before the requests are passed into the system interconnect. It is active for DMA only. Traffic in the other direction, from the system or PE to the device, is managed by other means – for example, the PE MMUs.

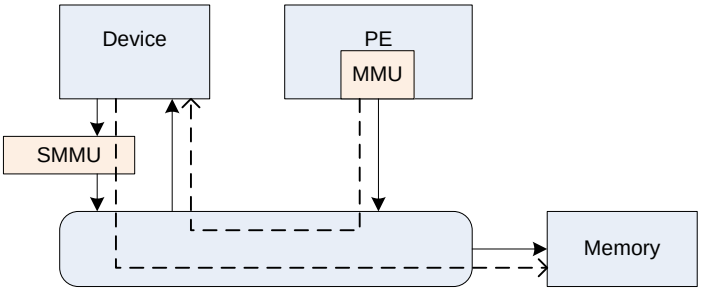


Figure 1: System MMU in DMA traffic

Translation of DMA addresses might be performed for reasons of isolation or convenience.

In order to associate device traffic with translations and to differentiate different devices behind an SMMU, requests have an extra property, alongside address, read/write, permissions, to identify a stream. Different streams are logically associated with different devices and the SMMU can perform different translations or checks for each stream. In systems with exactly one client device served by an SMMU the concept still stands, but might have only one stream.

A number of SMMUs might exist within a system. An SMMU might translate traffic from just one device or a set of devices.

The SMMU supports two stages of translation in a similar way to PEs supporting the Virtualization Extensions [4]. An incoming address is logically translated from VA to IPA in stage 1, then the IPA is input to stage 2 which translates the IPA to the output PA. Stage 1 is intended to be used by a software entity to provide isolation or translation to buffers within the entity, for example DMA isolation within an OS. Stage 2 is intended to be available in systems supporting the Virtualization Extensions and is intended to virtualize device DMA to guest VM address spaces.

2.1 History

- SMMUv1 supports a modest number of contexts/streams configured using registers, limiting scalability.
- SMMUv2 extends SMMUv1 with ARMv8-A translation table formats, large addresses, with the same limited number of contexts and streams.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

SMMUv1 and SMMUv2 map an incoming data stream onto one of many register-based context banks which indicate translation tables and translation configuration to use. The context bank might also indicate a second context bank for nested translation of a second stage (stage 1 and stage 2). The stream is identified using an externally-generated ID supplied with each transaction. A second ID might be supplied to determine the Security state of a stream or group of streams. The use of register-based configuration limits the number of context banks and support of thousands of concurrent contexts is not possible.

Because live data streams might potentially present transactions at any time, the available number of contexts limits the number of streams that might be concurrently enabled. For example, a system might have 1000 network interfaces that might all be idle but whose DMA might be triggered by incoming traffic at any time. The streams must be constantly available in order to function correctly. It is usually not possible to time-division multiplex a context between many devices requiring service.

2.2 SMMUv3.0 features

SMMUv3 provides feature to complement PCI Express [1] Root Complexes and other potentially large I/O systems by supporting large numbers of concurrent translation contexts.

- Memory-based configuration structures to support large numbers of streams.
- Implementations might support only stage 1, only stage 2 or both stages of translation. This capability, and other implementation-specific options, can be discovered from the register interface.
- Up to 16-bit ASIDs.
- Up to 16-bit VMIDs [5].
- Address translation and protection according to ARMv8.1 [5] Virtual Memory System Architecture [5]. SMMU translation tables shareable with PEs, allowing software the choice of sharing an existing table or creating an SMMU-private table.
- 49 bit VA (matching ARMv8-A's 2×48-bit translation table input sizes).

Support for the following is optional in an implementation:

- Either stage 1 or stage 2.
- Stage 1 and 2 support for the AArch32 (LPAE) and AArch64 translation table format.
- Secure stream support.
- Broadcast TLB invalidation.
- Hardware Translation Table Update (HTTU) of Access flag and Dirty state of a page. An implementation might support update of the Access flag only, update of both the Access flag and the Dirty state of the page, or no HTTU.
- PCIe ATS [3] and PRI, when used with compatible Root Complex.
- 16KB and 64KB page granules. However, the presence of 64KB page granules at both stage 1 and stage 2 is suggested to align with the PE requirements in the Server Base System Architecture.

Because the support of large numbers of streams using in-memory configuration causes the SMMUv3 programming interface to be significantly different to that of SMMUv2, SMMUv3 is not designed to be backward-compatible with SMMUv2.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

2.3 SMMUv3.1 features

SMMUv3.1 extends the base SMMUv3.0 architecture with the following features:

- Support for PEs implementing ARMv8.2-A:
  - Support for 52-bit VA, IPA and PA.
    - Note: An SMMUv3.1 implementation is not required to support 52-bit addressing, but the SMMUv3.1 architecture extends fields to allow an implementation the option of doing so.
  - Page-Based Hardware Attributes (PBHA).
  - EL0 vs EL1 execute never controls in stage 2 translation tables.
  - Note: ARMv8.2 introduces a Common not Private (CnP) concept to the PE which does not apply to the SMMU architecture, because all SMMU translations are treated as common.
- Support for transactions that perform cache-stash or destructive read side-effects.
- Performance Monitor Counter Group (PMCG) error status.

The SMMU programming interface register SMMU_AIDR indicates whether the SMMU implements SMMUv3.0 or SMMUv3.1, as follows:

- If SMMU_AIDR[7:0] == `0x0000`, the SMMU implements SMMUv3.0.
- If SMMU_AIDR[7:0] == `0x0001`, the SMMU implements SMMUv3.1.

Unless specified otherwise, all architecture behaviors apply equally to all minor revisions of SMMUv3.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

2.4 System placement

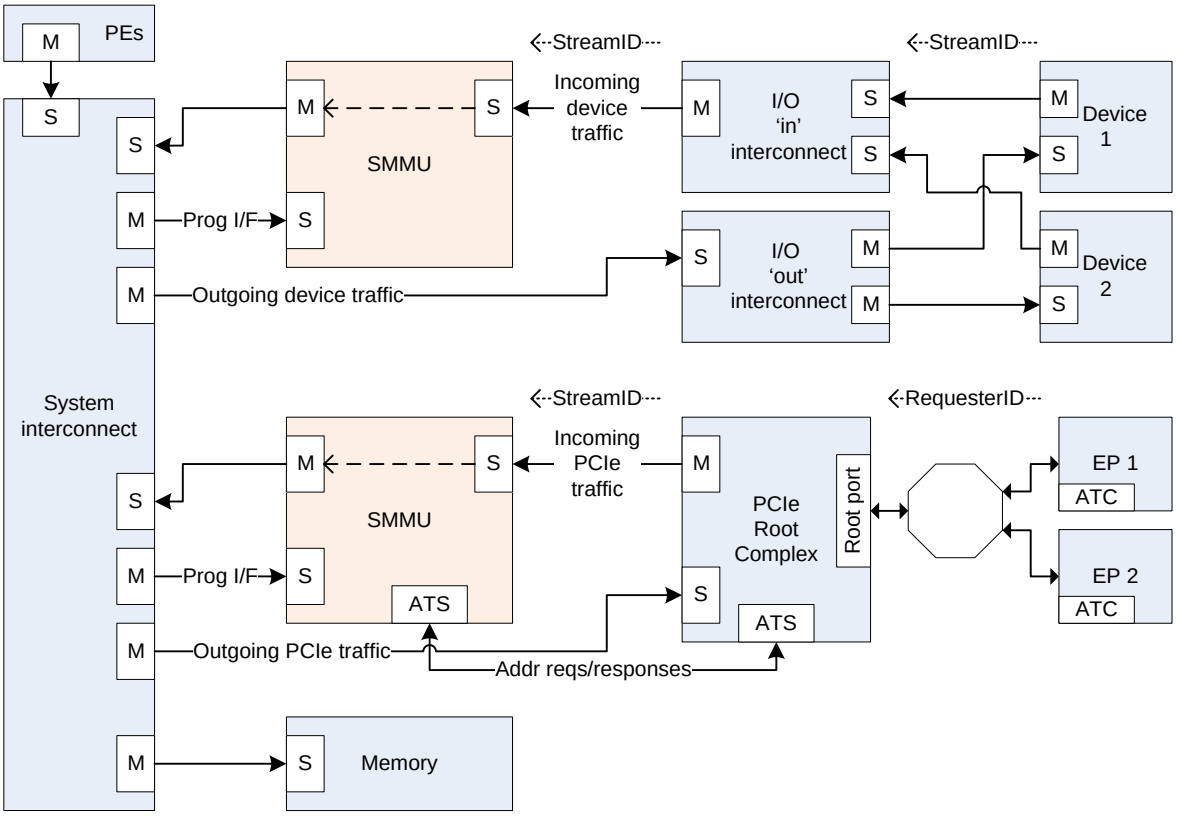


Figure 2: SMMU placement in an example system

Two example uses of an SMMU are shown in Figure 2. One SMMU interfaces incoming traffic from two client devices to the system interconnect. The devices can perform DMA using virtual, IPA or other bus address schemes and the SMMU translates these addresses to PAs. The second example SMMU interfaces one to one to a PCIe Root Complex (which itself hosts a network of endpoints). This illustrates an additional interface specified in this document, an ATS port to support PCIe ATS and PRI (or similar functionality for compatible non-PCIe devices).

Outgoing accesses to slave devices do not pass through an additional SMMU. In general, masters are behind an SMMU (or, in the case of PEs, have an inbuilt MMU), so outgoing accesses to slave devices are mediated by the MMU of the master. If a master has no MMU, it has full-system access. Therefore, its DMA must be mediated by software, and in this case only the most privileged system software can program it.

The SMMU has a programming interface that receives accesses from system software for setup and maintenance. The SMMU also makes accesses of its own (as a master) to configuration structures, for example to perform translation table walks. Whether the traffic originating from the SMMU itself shares the same interconnect resources as traffic passed through from device clients is implementation specific.

Each SMMU is configured separately to any others that might exist in the system.

Note: ARM recommends that SMMUs bridge I/O device DMA addresses onto system or physical addresses. ARM recommends that SMMUs are placed between a device master port (or I/O interconnect) and system

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

interconnect. Generally, ARM recommends that SMMUs are not placed in series and that the path of an SMMU to memory or other slave devices does not pass through another SMMU, whether for fetch of SMMU configuration data or client transactions.

Note: Interconnect-specific channels to support cache coherency are not shown in Figure 2.

The SMMU master interface to the system is intended to be IO-coherent, thereby providing IO-coherent access for the client devices of the SMMU. The SMMU slave interface for incoming device traffic does not require any coherency support. In addition, because there is no address translation in the outgoing direction, snoop traffic cannot be forwarded from the system towards the client devices so fully-coherent device caches cannot be placed behind an SMMU.

Note: It is feasible to implement an SMMU as part of a complex device containing fully-coherent caches in the same way that the MMU of a PE is paired to fully-coherent PE caches. Practically, this means the caches must be tagged with physical addresses.

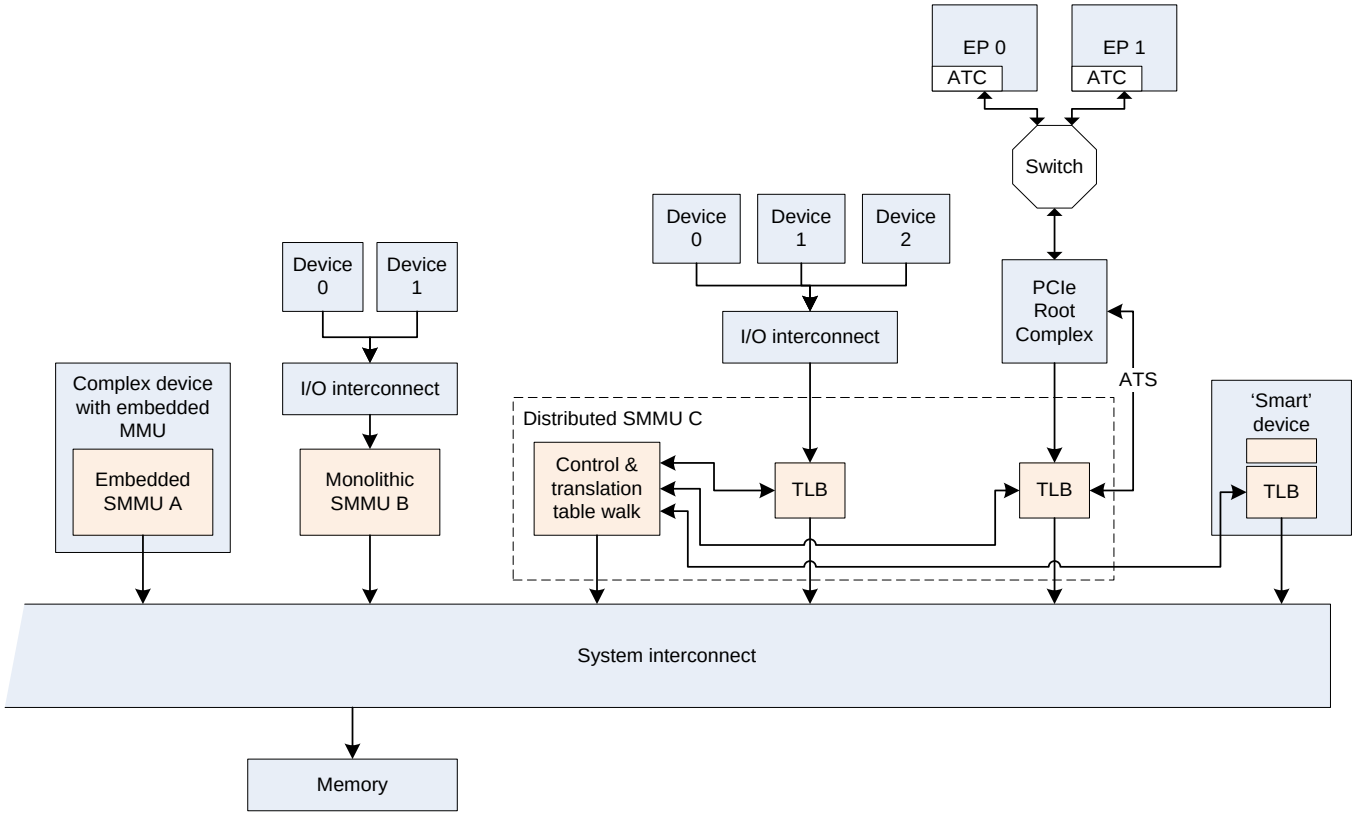


Figure 3: Example SMMU implementations

Figure 3 shows three example implementations of SMMU.

- SMMU A is implemented as part of a complex device, providing translation for accesses from that device only. ARM expects this implementation to have an SMMU programming interface in addition to device-specific control. This design can provide dedicated contention-free translation and TLBs.
- SMMU B is a monolithic block that combines translation, programming interface and translation table walk facilities. Two client devices use this SMMU as their path for DMA into the system.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- SMMU C is distributed and provides multiple paths into the system for higher bandwidth. It comprises:
    - A central translation table walker, which has its own master interface to fetch translation and configuration structures and queues and a slave interface to receive programming accesses. This unit might contain a macro-TLB and caches of configuration.
    - Remote TLB units which, on a miss, make translation requests to the central unit and cache the results locally. Two units are shown, supporting a set of three devices through one port, and a PCIe Root Complex through another.
        - The second TLB unit also provides an ATS interface to the Root Complex, so that the PCIe Endpoints can use ATS to make translation requests through to the central unit.
- Finally, a smart device is shown, which embeds a TLB and makes translation requests to the central unit of SMMU C. To software, this looks identical to a simple device connected behind a discrete TLB unit. This design provides a dedicated TLB for the device, but uses the programming interface and translation facilities of the central unit, reducing complexity of the device.

In all cases, it appears to software as though a device is connected behind a logically-separate SMMU (similar to Device 0/1 on SMMU B). All implementations give the illusion of simple read/write transactions arriving from a client device to a discrete SMMU, even if physically it is the device performing the read/write transactions directly into the system, using translations provided by an SMMU.

Note: This allows a single SMMU driver to be used for radically different SMMU implementations.

Note: Devices might integrate a TLB, or whole SMMU, for performance reasons, but a closely-coupled TLB might also be used to provide physical addresses suitable for fully-coherent device caches.

Regardless of the implementation style, this document uses the abstraction of client device transactions *arriving at an SMMU*. The boundary of SMMU might contain a single module or several distributed sub-components but these must all behave consistently.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3 Operation

3.1 Software interface

The SMMU has three interfaces that software uses:

1. Memory-based data structures to map devices to translation tables which are used to translate client device addresses.
2. Memory-based circular buffer queues. These are a Command queue for commands to the SMMU, an Event queue for event/fault reports from the SMMU, and a PRI queue for receipt of PCIe page requests.

   Note: The PRI queue is only present on SMMUs supporting PRI services. This additional queue allows processing of PRI requests from devices separate from event or fault reports.
3. A set of registers, some of which are secure-only, for discovery and SMMU-global configuration

The registers indicate the base addresses of the structures and queues, provide feature detection and identification registers and a global control register to enable queue processing and translation of traffic. When security is supported, an additional register set exists to allow Secure software to maintain Secure device structures, issue commands on a second Secure Command queue and read Secure events from a Secure Event queue.

In virtualization scenarios allowing stage 1 translation, a guest OS is presented with exactly the same programming interface and therefore believes it is in control of a real SMMU (albeit stage 1-only) with the same format of Command, Event, and optionally PRI, queues, and in-memory data structures.

Certain fields in architected SMMU registers and structures are marked as IMPLEMENTATION DEFINED. The content of these fields is specific to the SMMU implementation, but implementers must not use these fields in such a way that a generic SMMUv3 driver becomes unusable. Unless a driver has extended knowledge of particular IMPLEMENTATION DEFINED fields or features, the driver must treat all such fields as reserved and set them to 0. An implementation only uses IMPLEMENTATION DEFINED fields to enable extended functionality or features, and remains compatible with generic driver software by maintaining architected behavior when these fields are set to 0.

3.2 Stream numbering

An incoming transaction has an address, size, and attributes such as read/write, Secure/Non-secure, Shareability, Cacheability. If more than one client device uses the SMMU traffic must also have a sideband StreamID so the sources can be differentiated. How a StreamID is constructed and carried through the system is IMPLEMENTATION DEFINED. Logically, a StreamID corresponds to a device that initiated a transaction.

Note: The mapping of a physical device to StreamID must be described to system software.

ARM recommends that StreamID be a dense namespace starting at 0. The StreamID namespace is per-SMMU. Devices assigned the same StreamID but behind different SMMUs are seen to be different sources. A device might emit traffic with more than one StreamID, representing data streams differentiated by device-specific state.

StreamID is of IMPLEMENTATION DEFINED size, between 0 and 32 bits.

The StreamID is used to select a Stream Table Entry (STE) in a Stream table, which contains per-device configuration. The maximum size of in-memory configuration structures relates to the maximum StreamID span (see 3.3 below), with a maximum of $2^{\text{StreamIDSize}}$ entries in the Stream table.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Another property, SubstreamID, might optionally be provided to an SMMU implementing stage 1 translation. The SubstreamID is of IMPLEMENTATION DEFINED size, between 0 and 20 bits, and differentiates streams of traffic originating from the same logical block in order to associate different application address translations to each.

Note: An example would be a compute accelerator with 8 contexts that might each map to a different user process, but where the single device has common configuration meaning it must be assigned to a VM whole.

Note: The SubstreamID is equivalent to a PCIe PASID. Because the concept can be applied to non-PCIe systems, it has been given a more generic name in the SMMU. The maximum size of SubstreamID, 20 bits, matches the maximum size of a PCIe PASID.

The incoming transaction flags whether or not a SubstreamID is supplied and this might differ on a per-transaction basis.

Both of these properties and sizes are discoverable through the SMMU_IDR1 register. See section 16.4 for recommendations on StreamID and SubstreamID sizing.

The StreamID is the key that identifies all configuration for a transaction. A StreamID is configured to bypass or be subject to translation and such configuration determines which stage 1 or stage 2 translation to apply. The SubstreamID provides a modifier that selects between a set of stage 1 translations indicated by the StreamID but has no effect on the stage 2 translation which is selected by the StreamID only.

A stage 2-only implementation does not take a SubstreamID input. An implementation with stage 1 is not required to support substreams, therefore is not required to take a SubstreamID input.

The SMMU optionally supports two Security states and, if supported, the StreamID input to the SMMU is qualified by a SEC_SID flag that determines whether the input StreamID value refers to the Secure or Non-secure StreamID namespace. A Non-secure StreamID identifies an STE within the Non-secure Stream table and a Secure StreamID identifies an STE within the Secure Stream table. In this document, the term StreamID implicitly refers to the StreamID disambiguated by SEC_SID (if present) and does not refer solely to a literal StreamID input value (which would be associated with two STEs when security is supported) unless explicitly stated otherwise. See section 3.10.

ARM expects that, for PCI, StreamID is generated from the PCI RequesterID so that StreamID[15:0] == RequesterID[15:0]. When more than one Root Complex is hosted by one SMMU, ARM recommends that the 16-bit RequesterID namespaces are arranged into a larger StreamID namespace by using upper bits of StreamID to differentiate the contiguous RequesterID namespaces, so that StreamID[N:16] indicates which Root Complex (PCIe domain/segment) is the source of the stream source. In PCIe systems, the SubstreamID is intended to be directly provided from the PASID [2] in a one to one fashion.

Therefore, for SMMU implementations intended for use with PCI clients, supported StreamID size must be at least 16 bits.

3.3 Data structures and translation procedure

The SMMU uses a set of data structures in memory to locate translation data. Registers hold the base addresses of the initial root structure, the Stream Table. A Stream table entry (STE) contains stage 2 translation table base pointers, and also locates stage 1 configuration structures, which contain translation table base pointers. A Context descriptor (CD) represents stage 1 translation, and a Stream table entry represents stage 2 translation.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Therefore, there are two distinct groups of structures used by the SMMU:

- Configuration structures, which map from the StreamID of a transaction (a device originator identifier) to the translation table base pointers, configuration, and context under which the translation tables are accessed.
- Translation table structures that are used to perform the VA to IPA and IPA to PA translation of addresses for stage 1 and stage 2, respectively.

The procedure for translation of an incoming transaction is to first locate configuration appropriate for that transaction, (identified by its StreamID and, optionally, SubstreamID, and then to use that configuration to locate translations for the address used.

The first step in dealing with an incoming transaction is to locate the STE, which tells the SMMU what other configuration it requires.

Conceptually, an STE describes configuration for a client device in terms of whether it is subject to stage 1 or stage 2 translation or both. Multiple devices can be associated with a single Virtual Machine, so multiple STEs can share common stage 2 translation tables. Similarly, multiple devices (strictly, streams) might share common stage 1 configuration, therefore multiple STEs could share common CDs.

3.3.1 Stream Table lookup

The StreamID of an incoming transaction locates an STE. Two formats of Stream table are supported. The format is set by the Stream table base registers. The incoming StreamID is range-checked against the programmed table size, and a transaction is terminated if its StreamID would otherwise select an entry outside the configured Stream table extent (or outside a level 2 span). See section 6.3.24 and C_BAD_STREAMID. When two Security states are supported, as indicated by SMMU_S_IDR1.SECURE_IMPL == 1, a transaction is identified, using the SEC_SID flag, as associated with the Secure or Non-secure SMMU programming interfaces, see section 3.10.1. This determines whether the Secure or Non-secure Stream table (or caches of) is used.

3.3.1.1 Linear Stream Table

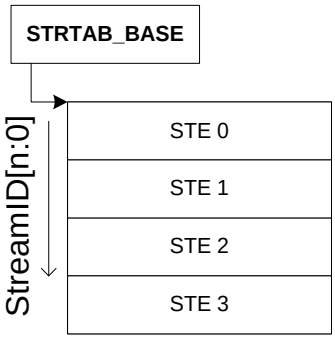


Figure 4: Linear Stream Table

A linear Stream table is a contiguous array of STEs, indexed from 0 by StreamID. The size is configurable as a 2^n multiple of STE size up to the maximum number of StreamID bits supported in hardware by the SMMU. The linear Stream Table format is supported by all SMMU implementations.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.3.1.2 2-level Stream Table

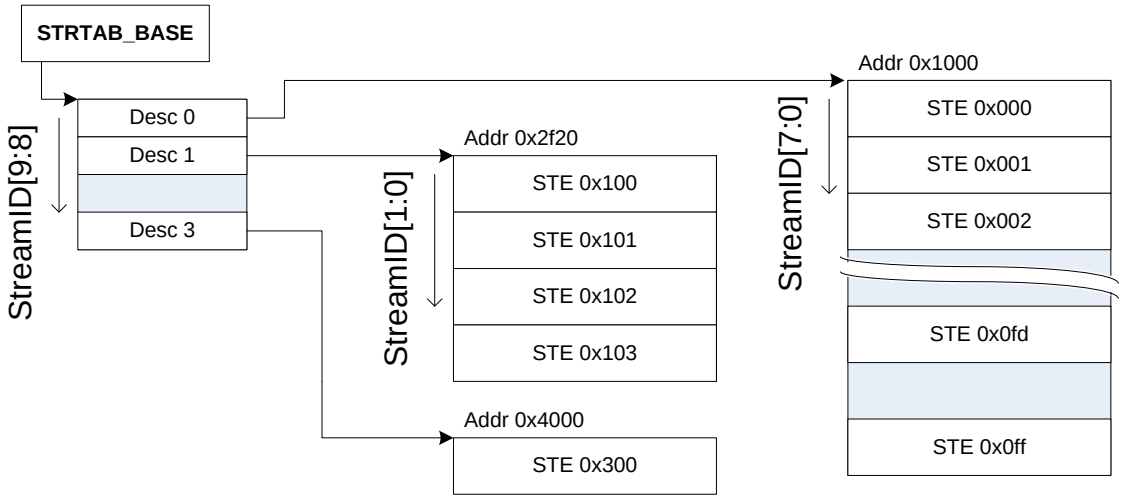


Figure 5: Example Two-level Stream Table with SPLIT == 8

A 2-level Stream table is a structure consisting of one top-level table that contains descriptors that point to multiple second-level tables that contain linear arrays of STEs. The span of StreamIDs covered by the entire structure is configurable up to the maximum number supported by the SMMU but the second-level tables do not have to be fully populated and might vary in size. This saves memory and avoids the requirement of large physically-contiguous allocations for very large StreamID spaces.

The top-level table is indexed by StreamID[n:SPT], where n is the uppermost StreamID bit covered, and SPT is a configurable Split point given by SMMU_(S_)STRTAB_BASE_CFG.SPLIT. The second-level tables are indexed by up to StreamID[SPT - 1:0], depending on the span of each table.

Support for the 2-level Stream table format is discoverable using the SMMU_IDR0.ST_LEVEL field. Where 2-level Stream Tables are supported, split points of 6, 8 and 10 bits can be used. Implementations support either a linear Stream Table format, or both linear and 2-level formats.

SMMUs supporting more than 64 StreamIDs (6 bits of StreamID) must also support 2-level Stream tables.

Note: Implementations supporting fewer than 64 StreamIDs might support 2-level Stream Tables, but doing so is not generally useful as all streams would fit within a single second-level table.

Note: This rule means that an implementation supports two-level tables when the maximum size of linear Stream table would be too big to fit in a 4KB page.

The top-level descriptors contain a pointer to the second-level table along with the StreamID span that the table represents. Each descriptor can also be marked as invalid.

This example top-level table is depicted in Figure 5, where the split point is set to 8:

Level 1 index	Valid	Level 2 pointer	Level 2 span

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

0	Y	0x1000	2^8
1	Y	0x2f20	2^2
2	N	–	–
3	Y	0x4000	2^0

In this example:

- StreamIDs 0-1023 (4 × 8-bit level 2 tables) are represented, though not all are valid.
- StreamIDs 0-255 are configured by the array of STEs at `0x1000` (each of which separately enables the relevant StreamID).
- StreamIDs 256-259 are configured by the array of STEs at `0x2F20`.
- StreamIDs 512-767 are all invalid.
- The STE of StreamID 768 is at `0x4000`.

A two-level table with a split point of 8 can reduce the memory usage compared to a large and sparse linear table used with PCIe. If the full 256 PCIe bus numbers are supported, the RequesterID or StreamID space is 16-bits. However, because there is usually one PCIe bus for each physical link and potentially one device for each bus, in the worst case a valid StreamID might only appear once every 256 StreamIDs.

Alternatively, a split point of 6 provides 64 bottom-level STEs, enabling use of a 4KB page for each bottom-level table.

3.3.2 StreamIDs to Context Descriptors

The STE contains the configuration for each stream indicating:

- Whether traffic from the device is enabled.
- Whether it is subject to stage 1 translation.
- Whether it is subject to stage 2 translation, and the relevant translation tables.
- Which data structures locate translation tables for stage 1.

If stage 1 is used, the STE indicates the address of one or more CDs in memory.

The CD associates the StreamID with stage 1 translation table base pointers (to translate VA into IPA), per-stream configuration, and ASID. If substreams are in use, multiple CDs indicate multiple stage 1 translations, one for each substream. Transactions provided with a SubstreamID are terminated when stage 1 translation is not enabled.

If stage 2 is used, the STE contains the stage 2 translation table base pointer (to translate IPA to PA) and VMID. If multiple devices are associated with a particular virtual machine, meaning they share stage 2 translation tables, then multiple STEs might map to one stage 2 translation table.

Note: ARM expects that, where hypervisor software is present, the Stream table and stage 2 translation table are managed by the hypervisor and the CDs and stage 1 translation tables associated with devices under guest control are managed by the guest OS. Additionally, the hypervisor can make use of separate hypervisor stage 1

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

translations for its own internal purposes. Where a hypervisor is not used, a bare-metal OS manages the Stream table and CDs. For more information, see section 3.6.

When a SubstreamID is supplied with a transaction and the configuration enables substreams, the SubstreamID indexes the CDs to select a stage 1 translation context. In this configuration, if a SubstreamID is not supplied, behavior depends on the STE.S1DSS flag:

- When STE.S1DSS == `0b00`, all traffic is expected to have a SubstreamID and the lack of SubstreamID is an error. A transaction without a SubstreamID is aborted and an event recorded.
- When STE.S1DSS == `0b01`, a transaction without a SubstreamID is accepted but is treated exactly as if its configuration were stage 1-bypass. The stage 1 translations are enabled only for transactions with SubstreamIDs.
- When STE.S1DSS == `0b10`, a transaction without a SubstreamID is accepted and uses the CD of Substream 0. Under this configuration, transactions that arrive with SubstreamID 0 are aborted and an event recorded.

The ASID and VMID values provided by the CD and STE structures tag TLB entries created from translation lookups performed through configuration from the CD and STEs. These tags are used on lookup to differentiate translation address spaces between different streams, or to match entries for invalidation on receipt of broadcast TLB maintenance operations. Implementations might also use these tags to efficiently allow sharing of identical translation tables between different streams.

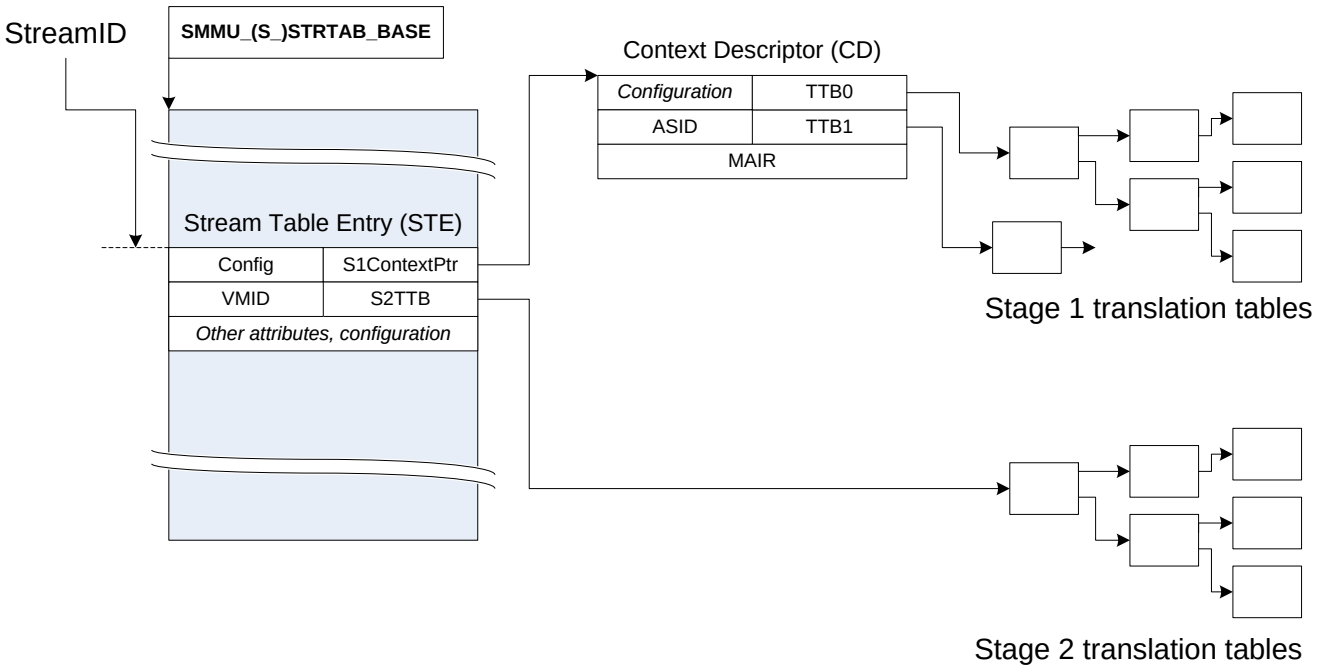


Figure 6: Configuration structure example

Figure 6 shows an example configuration in which a StreamID selects an STE from a linear Stream table, the STE points to a translation table for stage 2 and points to a single CD for stage 1 configuration, and then the CD points to translation tables for stage 1.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

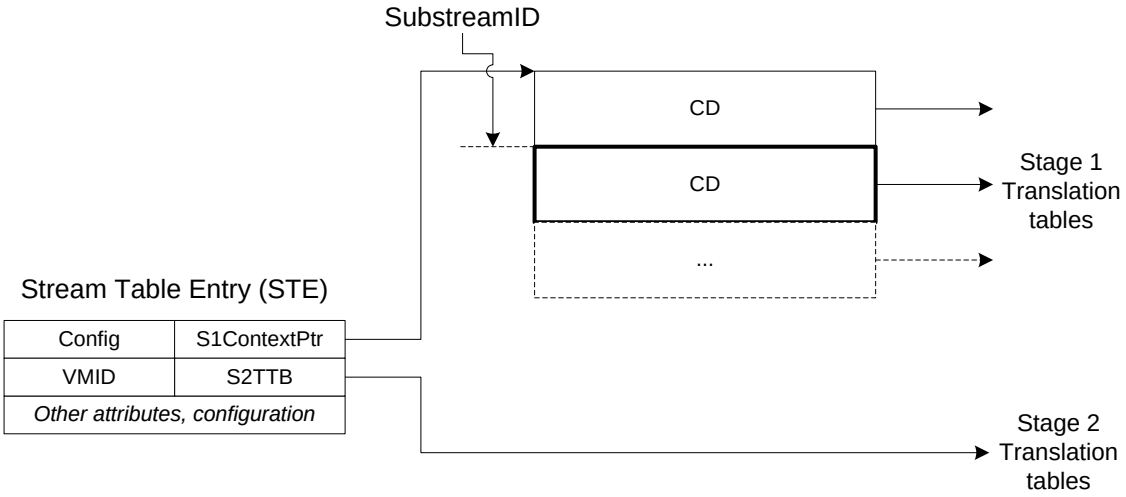


Figure 7: Multiple Context Descriptors for Substreams

Figure 7 shows a configuration in which an STE points to an array of several CDs. An incoming SubstreamID selects one of the CDs and therefore the SubstreamID determines which stage 1 translations are used by a transaction.

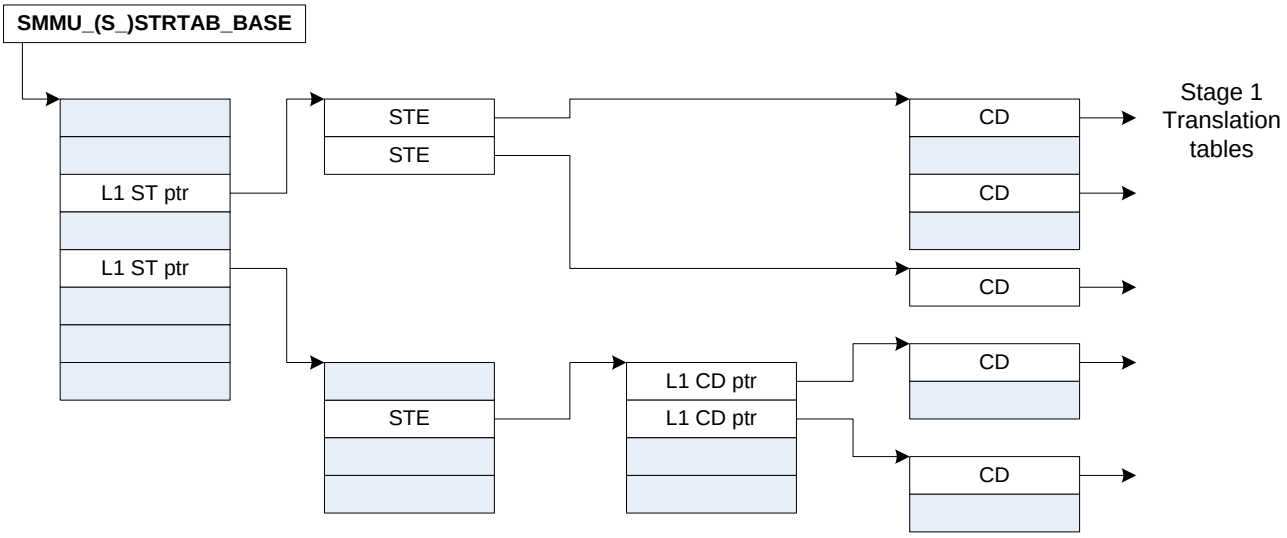


Figure 8: Multi-level Stream and CD tables

Figure 8 shows a more complex layout in which a multi-level Stream table is used. Two of the STEs point to a single CD, or a flat array of CDs, whereas the third STE points to a multi-level CD table. With multiple levels, many streams and many substreams might be supported without large physically-contiguous tables.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

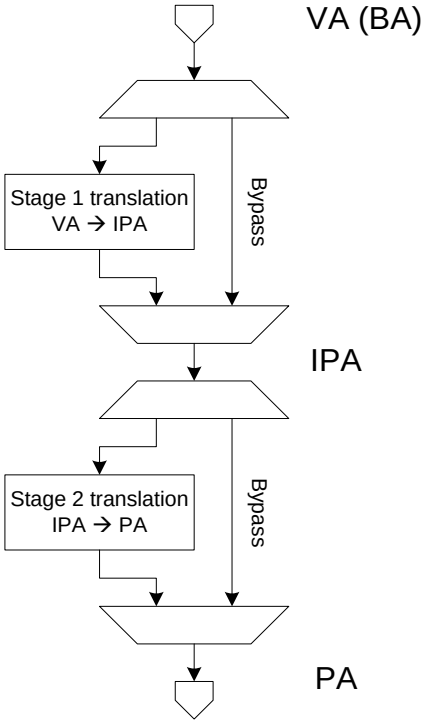


Figure 9: Translation stages and addresses

An incoming transaction is dealt with in a number of logical steps:

1. If the SMMU is globally disabled (for example when it has just come out of reset with SMMU_CR0.SMMUEN == 0), the transaction passes through the SMMU without any address modification. Global attributes, such as memory type or Shareability, might be applied from the SMMU_GBPA register of the SMMU. Or, the SMMU_GBPA register might be configured to abort all transactions.
2. If the global bypass described in (1) does not apply, the configuration is determined:
   a. An STE is located.
   b. If the STE enables stage 2 translation, the STE contains the stage 2 translation table base.
   c. If the STE enables stage 1 translation, a CD is located. If stage 2 translation is also enabled by the STE, the CD is fetched from IPA space which uses the stage 2 translations. Otherwise, the CD is fetched from PA space.
3. Translations are performed, if the configuration is valid.
   a. If stage 1 is configured to translate, the CD contains a translation table base which is walked. This might require stage 2 translations, if stage 2 is enabled for the STE. Otherwise, stage 1 bypasses translation and the input address is provided directly to stage 2.
   b. If stage 2 is configured to translate, the STE contains a translation table base that performs a nested walk of a stage 1 translation table if enabled, or a normal walk of an incoming IPA. Otherwise, stage 2 bypasses translation and the stage 2 input address is provided as the output address.
4. A transaction with a valid configuration that does not experience a fault on translation has the output address (and memory attributes, as appropriate) applied and is forwarded.

Note: This sequence illustrates the path of a transaction on a Non-secure stream. If two Security states are supported, the path of a transaction on a Secure stream is similar, except SMMU_S_CR0.SMMUEN and SMMU_S_GBPA control bypass, and stage 2 is not supported.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

An implementation might cache data as required for any of these steps. Section 16.2 describes caching of configuration and translation structures.

Furthermore, events might occur at several stages in the process that prevent the transaction from progressing any further. If a transaction fails to locate valid configuration or is of an unsupported type, it is terminated with an abort, and an event might be recorded. If the transaction progresses as far as translation, faults can arise at either stage of translation. The configuration that is specific to the CD and STEs that are used determines whether the transaction is terminated or whether it is stalled, pending software fault resolution, see section 3.12.

The two translation stages are described using the VA to IPA and IPA to PA stages of the ARMv8-A Virtualization terminology.

Note: Some systems refer to the SMMU input as a Bus Address (BA). The term VA emphasizes that the input address to the SMMU can potentially be from the same virtual address space as a PE process (using VAs).

Unless otherwise specified, translation tables and their configuration fields act exactly the same way as their equivalents specified in the ARMv8-A Translation System for PEs [4].

If an SMMU does not implement one of the two stages of translation, it behaves as though that stage is configured to permanently bypass translation. Other restrictions are also relevant, for example it is not valid to configure a non-present stage to translate. An SMMU must support at least one stage of translation.

3.3.3 Configuration and Translation lookup

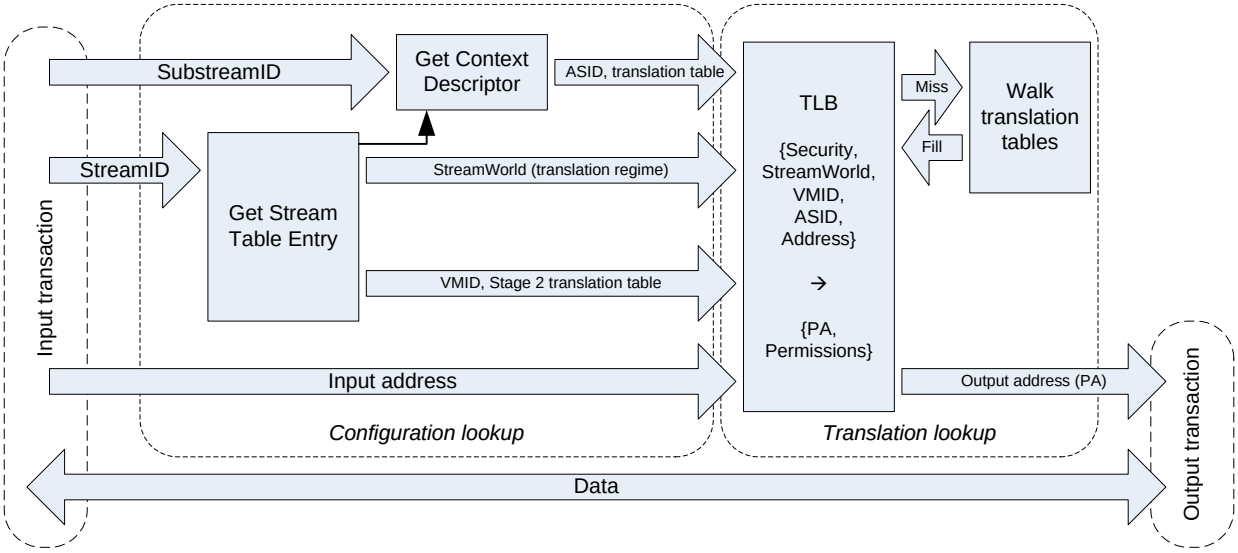


Figure 10: Configuration and translation lookup sequence

Figure 10 illustrates the concepts that are used in this document when referring to a *configuration lookup* and *translation lookup*.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

As described in 3.3.2 above, an incoming transaction is first subject to a configuration lookup, and the SMMU determines how to begin to translate the transaction. This involves locating the appropriate STE then, if required, a CD.

The configuration lookup stage does not depend on the input address and is a function of the:

- SMMU global register configuration.
- Incoming transaction StreamID.
- Incoming transaction SubstreamID (if supplied).

The result of the configuration lookup is the stream or substream-specific configuration that locates the translation, including:

- Stage 1 translation table base pointers, ASID, and properties modifying the interpretation or walk of the translation tables (such as translation granule).
- Stage 2 translation table base pointer, VMID and properties modifying the interpretation or walk of the translation table.
- Stream-specific properties, such as the StreamWorld (the Exception Level, or translation regime, in PE terms) to which the stream is assigned.

The translation lookup stage logically works the same way as a PE memory address translation system. The output is the final physical address provided to the system, which is a function of the:

- Input address.
- StreamWorld (Stream Security state and Exception level), ASID and VMID (which are provided from the previous step).

Figure 10 shows a PE-style TLB used in the translation lookup step. ARM expects the SMMU to use a TLB to cache translations instead of performing translation table walks for each transaction, but this is not mandatory.

Note: For clarity, Figure 10 does not show error reporting paths or CD fetch through stage 2 translation (which would also access the TLB or translation table walk facilities). An implementation might choose to flatten or combine some of the steps shown, while maintaining the same behavior.

A translation has a StreamWorld property that denotes the translation regime and is directly equivalent to an Exception level on a PE. All caches of translations are tagged with a StreamWorld which is matched on lookup and invalidation. The StreamWorld is defined by the configuration that inserts or looks up a translation. The StreamWorld is defined by the combination of the Security state of an STE, its STE.Config field, its STE.STRW field and SMMU_CR2.E2H. See the STE.STRW field in section 5.2. In addition to insertion and lookup, the StreamWorld/Exception level/translation regime affects the scope of different types of TLB invalidations. See section 3.17.

A translation is tagged with one of the following StreamWorlds:

NS-EL1	On a PE, equivalent to Non-secure EL1
EL2	Equivalent to EL2 when E2H == 0 (has no ASID)
EL2-E2H	Equivalent to EL2 when E2H == 1 (has ASID tag)

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Secure	Equivalent to either: • Secure EL1 and Secure EL3 when Secure EL3 is running in AArch32 (in the PE, a single translation regime is present for Secure). • Secure EL1 when Secure EL3 is running in AArch64 (EL3 has a separate translation regime). See section 3.10.2.
EL3	Equivalent to EL3 when EL3 is running in AArch64

Note: StreamWorld can differentiate multiple translation regimes in the SMMU that are associated with different bodies of software at different Exception levels. For example, a Secure Monitor EL3 translation for address `0x1000` is different to (and unaffected by) a hypervisor EL2 translation for address `0x1000`, as are NS-EL1 translations for address `0x1000`. In general, ARM expects that the StreamWorld configured for a stream in the SMMU will match the Exception level of the software that controls the stream or device.

In the same way as in an ARMv8-A MMU, a translation is architecturally unique if it is identified by a unique set of {StreamWorld, VMID, ASID, Address} input parameters.

For example, the following are unique and can all co-exist in a translation cache:

- Entries with the same address, but different ASIDs.
- Entries with the same address and ASID, but different VMIDs.
- Entries with the same address and ASID but a different StreamWorld.

Architecturally, a translation is not uniquely identified by a StreamID and SubstreamID. This results in two properties:

- A translation is not required to be unique for a set of transaction input parameters (StreamID, SubstreamID).
    - Two streams can be configured to use the same translation configuration and the resulting ASID/VMID from their configuration lookup will identify a single set of shared translation cache entries.
- Multiple StreamID/SubstreamID configurations that result in identical ASID/VMID/StreamWorld configuration must maintain exactly the same configuration where configuration can affect TLB lookup.
    - For example, two streams configured for a stage 1, NS-EL1 with ASID == 3 must both use the same translation table base addresses and translation granule.

In this document, the term *TLB* is used to mean the concept of a translation cache, indexed by StreamWorld/VMID/ASID and VA.

SMMU cache maintenance commands therefore fall into two groups:

- Configuration cache maintenance, acting upon StreamIDs and SubstreamIDs.
- Translation cache maintenance (or TLB maintenance), acting on addresses, ASIDs, VMIDs and StreamWorld.

The second set of commands directly matches broadcast TLB maintenance operations that might be available from PEs in some systems.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.3.4 Transaction attributes: incoming, two-stage translation and overrides

In addition to an address, size and read/write attributes, an incoming transaction might be presented to the SMMU with other attributes, such as an access type (for example Device, WB-cached Normal memory), Shareability (for example Outer Shareable), cache allocation hints, and permissions-related attributes, instruction/data, privileged/unprivileged, Secure/Non-secure. Some of these attributes are used to check the access against the page permissions that are determined from the translation tables. After passing through the SMMU, a transaction presented to the system might also have a set of attributes, which might have been affected by the SMMU.

The details of how input attributes affect the attributes output into the system, in combination with translation table attributes and other configuration, is described in detail in section 13.

The input attributes are conceptually provided from the system, either conveyed from a client device that defines the transaction attributes in a device-specific way, or set in a system-specific way by the interconnect before the transaction is input to the SMMU.

As an overview:

- Permission-related attributes (instruction/data, privileged/unprivileged) and read/write properties are used for checking against translation table permissions, which might deny the access. The permission-related attributes input into the SMMU might be overridden on a per-device basis before the permission checks are performed, using the INSTCFG, PRIVCFG and NSCFG STE fields. The SMMU might output these attributes.

Note: The overrides might be useful if a device is not able to express a particular kind of traffic.

- Other attributes (memory type, Shareability, cache hints) are intended to have an effect on the memory system rather than the SMMU, for example, control cache lookup for the transaction. The attributes output into the memory system are a function of the attributes specified by the translation table descriptors (at stage 1, stage 2, or stage 1 and stage 2) used to translate the input address. The SMMU might convey attributes input from a device through this process, so that the device might influence the final transaction access, and input attributes might be overridden on a per-device basis using the MTCFG/MemAttr, SHCFG, ALLOCCFG STE fields. The input attribute, modified by these fields, is primarily useful for setting the resulting output access attribute when both stage 1 and stage 2 translation is bypassed (no translation table descriptors to determine attribute) but can also be useful for stage 2-only configurations in which a device stream might have finer knowledge about the required access behavior than the general virtual machine-global stage 2 translation tables.

The STE attribute and permission override fields, MTCFG/MemAttr, SHCFG, ALLOCCFG, INSTCFG, PRIVCFG and NSCFG, allow an incoming value to be used or, for each field, a specific override value to be selected. For example, INSTCFG can configure a stream as Always Data, replacing an incoming INST property that might be in either state. However, in SMMU implementations that are closely-coupled to, or embedded in, a device, the incoming attribute can always be considered to be the most appropriate. When an SMMU and device guarantee that the incoming attributes are correct, it is permissible for an SMMU to always use the incoming value for each attribute value. See SMMU_IDR1.ATTR_TYPES_OVR and SMMU_IDR1.ATTR_PERMS_OVR for more information. For an SMMU that cannot guarantee that these attributes are always provided correctly from the client device, for example a discrete SMMU design, ARM strongly recommends supporting overrides of incoming attributes.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.3.5 Translation table descriptors

The ARMv8-A VMSA defines bits [63:60] of stage 2 block and page descriptors as being reserved for use by a System MMU. In SMMUv3.1, these bits are Reserved, RES0.

Note: When PBHA is enabled for a bit in this range, bits [62:60] are affected by the PBHA mechanism. When PBHA is not enabled, the previous definition applies.

3.4 Address sizes

There are three address size concepts to consider in the SMMU, the input address size from the system, the Intermediate Address Size (IAS), and the Output Address Size (OAS):

- The SMMU input address size is 64 bits.
  - Note: See section 3.4.1 for recommendations on how a smaller interconnect or device address capability is presented to the SMMU.
- IAS reflects the maximum usable IPA of an implementation that is generated by stage 1 and input to stage 2:
  - This term is defined to illustrate the handling of intermediate addresses in this section and is not a configurable parameter.
  - The maximum usable IPA size of an SMMU is defined in terms of other SMMU implementation choices, as:

    ```
    IAS = MAX((SMMU_IDR0.TTF[0] == 1 ? 40 : 0), (SMMU_IDR0.TTF[1] == 1 ? OAS : 0));
    ```
  - An IPA of 40 bits is required to support of AArch32 LPAE translations, and AArch64 limits the maximum IPA size to the maximum PA size. Otherwise, when AArch32 LPAE is not implemented, the IPA size equals OAS, the PA size, and might be smaller than 40 bits.
  - The purpose of definition of the IAS term is to abstract away from these implementation variables.
- OAS reflects the maximum usable PA output from the last stage of AArch64 translations, and must match the system physical address size. The OAS is discoverable from SMMU_IDR5.OAS. Final-stage AArch32 translations always output 40 bits which are zero-extended into a larger OAS, or truncated to a smaller OAS.

Note: Except where explicitly noted, all address translation and fault checking behavior is consistent with ARMv8-A [4].

If the SMMU is disabled (with SMMU_(S_)CR0.SMMUEN == 0, and SMMU_(S_)GBPA.ABORT == 0 allows traffic bypass), the input address is presented directly to the output PA. If the input address of a transaction exceeds the size of the OAS, the transaction is terminated with an abort and no event is recorded. Otherwise, when SMMU_(S_)CR0.SMMUEN == 1, transactions are treated as described in the rest of this section.

When a stream selects an STE with STE.Config[2:0] == `0b100`, transactions bypass all stages of translation. If the input address of a transaction exceeds the size of the OAS, the transaction is terminated with an abort and a stage 1 Address Size fault (F_ADDR_SIZE) is recorded.

Note: In ARMv8-A PEs, when both stages of translation bypass, a (stage 1) Address Size fault might be generated where an (input) address is greater than the PA size, depending on whether a PE is in AArch32 or AArch64 state. This behavior does not directly translate to the SMMU because no configuration is available to select translation system when in bypass or disabled, therefore the address size is always tested.

When a stream selects an STE with one or more stages of translation present:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

For input to stage 1, the input address is treated as a VA (see section 3.4.1) and if stage 1 is not bypassed the following stage 1 address checks are performed:

1. On input, a stage 1 Translation fault (F_TRANSLATION) occurs if the VA is outside the range specified by the relevant CD:
    a. For a CD configured as AArch32 LPAE, the maximum input range is fixed at 32 bits, and the range of the address input into a given TTB0 or TTB1 translation table is determined by the T0SZ and T1SZ fields.

       Note: The arrangement of the TTB0/TTB1 translation table input spans might be such that there is a range of 32-bit addresses that is outside both of the TTB0 and TTB1 spans and will always cause a Translation fault.
    b. For a CD configured as AArch64, the range is determined by the T0SZ and T1SZ fields.
        i. For SMMUv3.0, up to a maximum of 49 bits (two 48-bit TTB0/TTB1).
        ii. For SMMUv3.1, when SMMU_IDR5.VAX == 1, each TTBx that is configured for a 64KB granule using CD.TGx has a maximum input size of 52 bits. When SMMU_IDR5.VAX == 0 or a TTBx is configured for a 4KB or 16KB granule, the maximum input size of the TTBx is 48 bits.

   A VA is inside the range only if it is correctly sign-extended from the top bit of the range size upwards, although an exception is made for Top Byte Ignore (TBI) configurations.

   Note: For example, with a 49-bit VA range and TBI disabled, addresses `0x0000FFFFFFFFFFFF` and `0xFFFF000000000000` are within the range but `0x0001000000000000` and `0xFFFE000000000000` are not. See 3.4.1 below for details.
2. The address output from the translation causes a stage 1 Address Size fault if it exceeds the range of the effective IPA size for the given CD:
    a. For AArch32 LPAE CDs, the IPA size is fixed at 40 bits (the IPS field of the CD is IGNORED).
    b. For AArch64 CDs, the IPA size is given by the effective value of the IPS field of the CD, which is capped to the OAS.

If bypassing stage 1 (because STE.Config[0] == 0, STE.S1DSS == `0b01` or if unimplemented), the input address is passed directly to stage 2 as the IPA. If the input address of a transaction exceeds the size of the IAS, a stage 1 Address Size fault occurs, the transaction is terminated with an abort and F_ADDR_SIZE is recorded. Otherwise, the address might still lie outside the range that stage 2 will accept. In this case, the stage 2 check 1 described in this section causes a stage 2 Translation fault.

Note: The TBI configuration can only be enabled when a CD is used (that is when stage 1 translates) and is always disabled when stage 1 is bypassed or disabled.

Note: The SMMU stage 1 bypass behavior is analogous to a PE with stage 1 disabled but stage 2 translating. The SMMU checks stage 1 bypassed addresses against the IAS, which (when AArch32 LPAE support is implemented) might be greater than the PA. This supports stage 2-only assignment of devices to guest VMs expecting to program 40-bit DMA addresses, which are input to stage 2 translation.

Note: This also means that an SMMU implementing only stage 2, or implementing both stages but translating through stage 2 only, can still produce a fault marked as coming from stage 1.

Stage 2 receives an IPA, and if not bypassing, the following stage 2 address size checks are performed:

1. On input, a stage 2 Translation fault occurs if the IPA is outside the range configured by the relevant S2T0SZ field of the STE.
    a. For an STE configured as AArch32 LPAE (see STE.S2AA64), the input range is capped at 40 bits (and cannot exceed 40 bits) regardless of the IAS size.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

b. For a STE configured as AArch64, the input range is capped by the IAS.

c. For SMMUv3.1, when OAS == IAS == 52, the stage 2 input range is further limited to 48 bits unless STE.S2TG indicates a 64KB granule.

   Note: This ensures, for a system having OAS < 40, that an AArch64 stage 2 can accept a 40-bit IPA from an AArch32 stage 1, if the SMMU supports AArch32.

2. The address output from the translation causes a stage 2 Address Size fault if it exceeds the effective PA output range:

   a. For an AArch64 STE, this is the effective value configured in the S2PS field of the STE (which is capped to the OAS).

      Note: For SMMUv3.1, the effective size can be 52 only if OAS == 52 and 64KB granules are used.

   b. For an AArch32 STE, this output range is fixed at 40 bits and the STE.S2PS field is IGNORED. If the OAS is less than 40, and if the output address is outside the range of the OAS, the address is silently truncated to fit the OAS.

   After this check, if the output address of stage 2 is smaller than the OAS, the address is zero-extended to match the OAS.

If bypassing stage 2 (because STE.Config[1] == 0 or if unimplemented), the IPA is presented directly as the PA output address. If the IPA is outside the range of the OAS, the address is silently truncated to fit the OAS. If the IPA is smaller than the OAS, it is zero-extended.

Note: Because the SMMU contains configuration structures that are checked for validity before beginning translation table walks, certain configuration errors are detected as an invalid structure configuration. This includes STE.S2TTB being out of range of the effective stage 2 output address size, or CD.TTBx being out of range of the effective stage 1 output address size. These invoke C_BAD_STE or C_BAD_CD configuration faults, respectively, instead of an Address Size fault.

3.4.1 Input address size and Virtual Address size

The architectural input address size of the SMMU is 64 bits. If a client device outputs an address smaller than 64 bits, or if the interconnect between a client device and the SMMU input supports fewer than 64 bits of address, the smaller address is converted to a 64-bit SMMU input address in a system-specific manner. This conversion is outside of the scope of this specification.

ARMv8.0 [4] and ARMv8.1 [5] support a maximum of a 49-bit VA in AArch64, meaning there are up to 49 significant lower bits sign-extended to a 64-bit address. ARMv8.2 supports a maximum of a 53-bit VA or 49-bit VA in AArch64, meaning there are up to 53 or 49 significant lower bits sign-extended to a 64-bit address. Stage 1 translation contexts configured as AArch64 have input VA ranges configurable up to the maximum (arranged as two halves and translated through TTB0 and TTB1).

The term *VAS* represents the VA size chosen for a given SMMU implementation. When SMMU_IDR5.VAX == 0, this is 49 bits (2 x 48 bits). When SMMU_IDR5.VAX == 1, this is 53 bits (2 x 52 bits).

Note: In SMMUv3.0, SMMU_IDR5.VAX is RES0 and therefore VAS is always 49 bits.

Stage 1's high translation table, TTB1, can only be selected if VAS significant bits of address are presented to the SMMU sign-extended. If applications require use of both TTB0 and TTB1 then the system design must transmit addresses of at least VAS bits end-to-end, from device address registers through the interconnect to the SMMU, and that sign-extension occurs from the input MSB upwards as described in this section.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Stage 1 translation contexts configured as AArch32 have a 32-bit VA. In this case, bits [31:0] of the input address are used directly as the VA. A Translation fault is raised if the upper 32 bits of the input address are not all zero. The TxSZ fields are used to select TTB0 or TTB1 from the upper bits [31:n] in the input range.

Input range checks made for a stage 1 AArch64 translation table configured (with TxSZ) for an input range of N significant bits fail unless bits VA[*AddrTop*:N - 1] are identical.

- When Top Byte Ignore (TBI) is not used, *AddrTop* == 63.
- When Top Byte Ignore is enabled, *AddrTop* == 55, which means that VA[63:56] are ignored.

When TBI is enabled, only VA[55:N-1] must be identical and the effective VA[63:56] bits are taken to be a sign-extension of VA[55] for translation purposes.

Note: TBI configuration is part of the CD, so it can only be enabled when stage 1 translates. When stage 1 is bypassed or disabled, no CD is used and TBI is always disabled.

The term *Upstream Address Size* (UAS) represents the number of effective bits of address presented to the SMMU from a client device:

1. If 57 ≤UAS < 64, TBI has meaning as there are bits supplied in VA[63:56] that might differ from VA[55:(VAS-1)]. TBI determines whether a Translation fault is invoked if they differ.
2. If VAS ≤UAS < 57, TBI is meaningless as the input sign-extension means VA[63:56] cannot differ from VA[55].
3. If UAS ≤VAS, the range checks can only fail if translation table range is configured with a T0SZ, or T1SZ, if UAS == 49, smaller than the presented address. That is, the maximum configuration of stage 1 translation tables covers any presented input address.

For AArch64, the stage 1 translation table, TTB0 or TTB1, is selected from the uppermost address bit. This is given by VA[63] when TBI == 0, or by VA[55] when TBI == 1. Therefore, because an address size from the client device that is less than or equal to the VAS bits is zero-extended to 64, this means VA[63] == VA[55] == 0 and TTB1 is never selected.

If any upper address bits of a 64-bit address programmed into a peripheral are not available to the SMMU sign-checking logic, whether by truncation in the interconnect or peripheral, software must not rely on mis-programmed upper bits to cause a Translation fault in the SMMU. If such checking is required within such a system, software must check the validity of upper bits of DMA addresses programmed into such a device.

All input address bits are recorded unmodified in SMMU fault event records.

3.4.2 Address alignment checks

The SMMU architecture does not check the alignment of incoming transaction addresses.

Note: For a PE, the alignment check is based on the size of an access. This semantic is not directly applicable to client device accesses.

3.4.3 Address sizes of SMMU-originated accesses

Distinct from client device accesses forwarded into the system, the SMMU originates accesses to the system for the purposes of:

- Configuration structure access (STE, CD).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Queue access (Command, Event, PRI).
- MSI interrupt writes.
- Last-stage translation table walks:
    - Note: Addresses output from stage 1 walks in a nested configuration are input to stage 2 and translated in the expected manner (including causing stage 1 Address Size faults, or stage 2 Translation faults from IPAs outside the stage 2 translation range), rather than being output into the system directly.

An access address can be out of range if it relates to a base address that is already greater than an allowed address size, or if an index is applied to a base address so that the result is greater than an allowed address size. If an access address is calculated to be a PA value greater than the SMMU OAS or physical address size, or an IPA value greater than the IAS or intermediate address size, behavior is as follows:

Access type	Configured by	Address type	Behavior when address too large
CD fetch or L1CD fetch	STE.S1ContextPtr	If stage 1-only, PA	Truncate to OAS or F_CD_FETCH or C_BAD_STE (1)
		If stage 2 present, IPA	Truncate to IAS or C_BAD_STE or Stage 2 Translation fault (2)
CD fetch	L1CD.L2Ptr	If stage 1-only, PA	Truncate to OAS or F_CD_FETCH or C_BAD_SUBSTREAMID (3)
		If stage 2 present, IPA	Truncate to IAS or C_BAD_SUBSTREAMID or stage 2 Translation fault (4)
STE fetch	SMMU_(S_)STRTAB_BASE or L1STD.L2Ptr	PA	Truncate to OAS or F_STE_FETCH (5)
Queue access	SMMU_(S_)*Q_BASE	PA	Truncate to OAS (6)
MSI write	SMMU_(S_)*_IRQ_CFG{0,1,2} or CMD_SYNC arguments	PA	Truncate to OAS (6)
Last-stage translation table walk	Addresses derived from intermediate translation table descriptors located using STE.S2TTB or CD.TTB{0,1}, after the first level TTD fetch.	PA	Stage 1/2 Address Size fault
	Starting-level TTD address in STE.S2TTB or CD.TTB{0,1},	PA	CD or STE ILLEGAL (see CD.TTB{0,1} and STE.S2TTB description)

In the context of these respective access types:

1. An SMMUv3.1 implementation generates C_BAD_STE and terminates the transaction. It is CONSTRAINED UNPREDICTABLE whether an SMMUv3.0 implementation:
    a. Generates an F_CD_FETCH and terminates the transaction. The event contains the non-truncated fetch address.
    b. Generates a C_BAD_STE and terminates the transaction.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

   c. Truncates STE.S1ContextPtr to the OAS and initiates a read of a CD/L1CD from this address (translation continues).

2. It is CONSTRAINED UNPREDICTABLE whether an implementation:
   a. Generates a C_BAD_STE and terminates the transaction.
   b. Inputs the IPA to stage 2 without truncation, generating a stage 2 Translation fault that reports a non-truncated fault address.
   c. SMMUv3.0 only, inputs the IPA to stage 2 with truncation to the IAS. If translation is successful, initiates a read of a CD/L1CD from the result otherwise generates a stage 2 fault that reports a truncated fault address.
3. An SMMUv3.1 implementation generates C_BAD_SUBSTREAMID and terminates the transaction. It is CONSTRAINED UNPREDICTABLE whether an SMMUv3.0 implementation:
   a. Generates an F_CD_FETCH and terminates the transaction. The event contains the non-truncated fetch address.
   b. Generates a C_BAD_SUBSTREAMID and terminates the transaction.
   c. Truncates L1CD.L2Ptr to the OAS and initiates a read of a CD from this address (translation continues).
4. It is CONSTRAINED UNPREDICTABLE whether an implementation:
   a. Generates a C_BAD_SUBSTREAMID and terminates the transaction.
   b. Inputs the IPA to stage 2 without truncation, generating a stage 2 Translation fault that reports a non-truncated fault address.
   c. SMMUv3.0 only, inputs the IPA to stage 2 with truncation to the IAS. If translation is successful, initiates a read of a CD from the result otherwise generates a stage 2 fault that reports a truncated fault address.
5. It is CONSTRAINED UNPREDICTABLE whether an implementation truncates an STE fetch address (and continues translation) or generates an F_STE_FETCH condition which terminates the transaction and might deliver an error event.
6. Note: When hypervisor software presents an emulated SMMU interface to a guest, ARM recommends that guest-provided addresses are correctly masked to the IPA size to ensure consistent SMMU behavior from the perspective of the guest driver.

In all cases where a non-truncated address is reported in a fault (for instance, a stage 2 Translation fault), the reported address is the calculated address of the structure being accessed, for example an L1CD address calculated from a base address of STE.S1ContextPtr indexed by the incoming SubstreamID to locate a L1CD structure.

The address of an L1CD or CD, given by STE.S1ContextPtr or L1CD.L2Ptr, is not subject to a stage 1 Address Size fault check.

In summary, configuration registers, command fields and structure fields programmed with out-of-range physical addresses might truncate the addresses to the OAS or PA size.

Note: This behavior in part arises from the fact that register address fields are not required to provide storage for high-order physical address bits beyond the OAS. See section 6.3 for details.

Note: Commands, register and structure fields taking IPA addresses store the entire field width so that a potential stage 2 fault can be correctly raised (providing a full non-truncated IPA in a fault record).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.5 Command and Event queues

All SMMU queues for both input to, and output from the SMMU are arranged as circular buffers in memory. A programming interface has one Command queue for input and one Event queue (and optionally one PRI queue) for output. Each queue is used in a producer-consumer fashion so that an output queue contains data produced by the SMMU and consumed by software. An input queue contains data produced by software, consumed by the SMMU.

3.5.1 SMMU circular queues

A queue is arranged as a 2^n-items sized circular FIFO with a base pointer and two index registers, PROD and CONS indicating the producer and consumer current positions in the queue. In each of the output and input roles, only one index is maintained by the SMMU, with the other is maintained by software.

For an input queue (Command queue), the PROD index is updated by software after inserting an item into the queue, and is read by the SMMU to determine new items. The CONS index is updated by the SMMU as items are consumed, and is read by software to determine that items are consumed and space is free. An output queue is the exact opposite.

PROD indicates the index of the location that can be written next, if the queue is not full, by the producer. CONS indicates the index of the location that can be read next, if the queue is not empty. The indexes must always increment and wrap to the bottom when they pass the top entry of the queue.

The queues use the mirrored circular buffer arrangement that allows all entries to be valid simultaneously (rather than N-1 valid entries in other circular buffer schemes). Each index has a wrap flag, represented by the next higher bit adjacent to the index value contained in PROD and CONS. This bit must toggle each time the index wraps off the high end and back onto the low end of the buffer. It is the responsibility of the owner of each index, producer or consumer, to toggle this bit when the owner updates the index after wrapping. It is intended that software reads the register, increments or wraps the index (toggling wrap when required) and writes back both wrap and index fields at the same time. This single update prevents inconsistency between index and wrap state.

- If the two indexes are equal and their wrap bits are equal, the queue is empty and nothing can be consumed from it.
- If the two indexes are equal and their wrap bits are different, the queue is full and nothing can be produced to it.
- If the two indexes differ or the wrap bits differ, the consumer consumes entries, incrementing the CONS index until the queue is empty (both indices and wrap bits are equal).

Therefore, the wrap bits differentiate the cases of an empty buffer and a full buffer where otherwise both indexes would indicate the same location in both full and empty cases.

On initialization, the queue indexes are written by the agent controlling the SMMU before enabling the queue. The queue indexes must be initialized into one of the following consistent states:

- PROD.WR == CONS.RD and PROD.WR_WRAP == CONS.RD_WRAP, representing an empty queue.
    - Note: ARM expects this to be the state on normal initialization.
- PROD.WR == CONS.RD and PROD.WR_WRAP != CONS.RD_WRAP, representing a full queue.
- PROD.WR > CONS.RD and PROD.WR_WRAP == CONS.RD_WRAP, representing a partially-full queue.
- PROD.WR < CONS.RD and PROD.WR_WRAP != CONS.RD_WRAP, representing a partially-full queue.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The agent controlling the SMMU must not write queue indexes to any of the following inconsistent states whether at initialization or after the queue is enabled:

- PROD.WR > CONS.RD and PROD.WR_WRAP != CONS.RD_WRAP
- PROD.WR < CONS.RD and PROD.WR_WRAP == CONS.RD_WRAP

If the queue indexes are written to an inconsistent state, one of the following CONSTRAINED UNPREDICTABLE behaviors is permitted:

- The SMMU consumes, or produces as appropriate to the given queue, queue entries at UNKNOWN locations in the queue.
- The SMMU does not consume, or produce as appropriate, queue entries while the queue indexes are in an inconsistent state.

Each circular buffer is 2^n-items in size, where $0 \leq n \leq 19$. An implementation might support fewer than 19 bits of index. Each PROD and CONS register is 20 bits in size to accommodate the maximum 19-bit index plus the wrap bit. The actual buffer size is determined by software, up to the discoverable SMMU implementation-defined limit. The position of the wrap bit depends on the configured index size.

Note: For example, when a queue is configured with 128 entries it means:

- The queue indices are 7-bit.
- PROD.WR and CONS.RD fields are 7 bits large. The queue indexes are bits [6:0] of PROD and CONS.
- The wrap bit [7] of PROD and CONS registers. Bits [19:8] are ignored.

The lifecycle of a circular buffer is shown in Figure 11.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

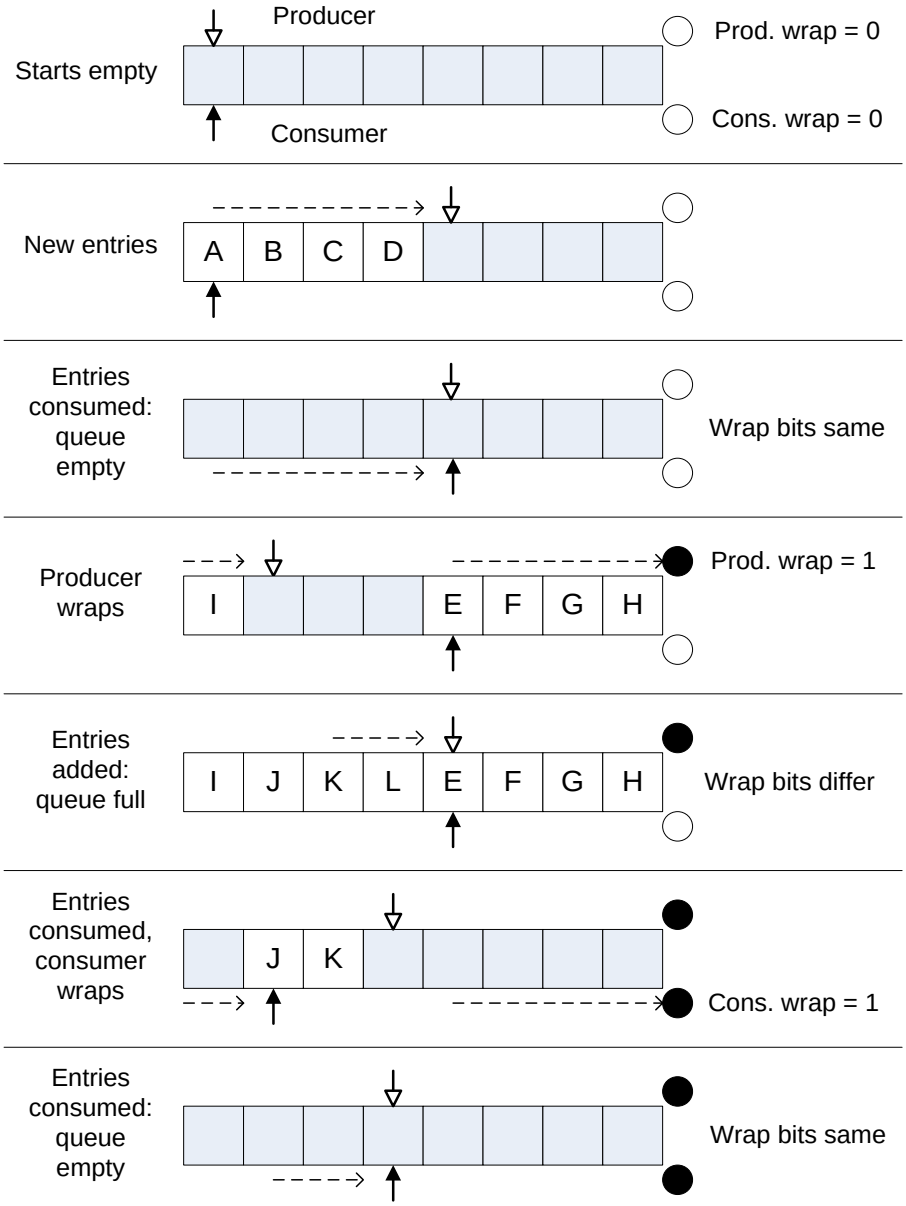


Figure 11: Circular buffer/queue operation

When producing or consuming entries, software must only increment an index (except when an increment will cause a wrap to the start). The index must never otherwise be moved backwards. The SMMU makes the same guarantee, only incrementing or wrapping its index values.

There is one Command queue per implemented Security state. The SMMU commands are consumed in order from this queue.

The Event queues receive asynchronous events such as faults recorded from device traffic or configuration errors (which might be discovered only when device traffic causes the SMMU to traverse the structures). On the Non-secure side, there is one global Event queue which receives events from all Non-secure streams and configuration.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When SMMU_S_IDR1.SECURE_IMPL == 1, there is also one Secure Event queue which receives events from all Secure streams and configuration, see section 3.10.2.

All output queues are appended to sequentially.

3.5.2 Queue entry visibility semantics

Any producer (whether the SMMU or software) must ensure that if an update to the PROD index value is observable by the consumer, all new queue entries are observable by the consumer. For output queues from the SMMU (Event and PRI queues), the SMMU writes queue data to memory and, when that data becomes visible with respect to the rest of the Shareability domain, the SMMU allows the updated PROD index value to be observed. This is the first point that a new queue entry is visible to the consumer.

A consumer must not assume presence of a new valid entry in a queue through any mechanism other than having first observed an updated PROD index that covers the entry position. If a consumer reads a queue entry beyond the point indicated by the last read of the PROD index, the entry contains UNKNOWN data.

Note: Interrupt ordering rules also exist, see section 3.18. The SMMU makes queue updates observable through the PROD index no later than at the point where it asserts the queue interrupt.

Note: Software must not assume a new queue item is present when an interrupt arrives, without first reading the PROD index. If, for example, a prior interrupt handler consumed all events including those of a second batch (with a second interrupt), the next interrupt handler invocation might find no new queue entries.

3.5.3 Event queue behavior

The SMMU might support configurable behavior on *Translation-related* faults, which enable a faulting transaction to be *stalled*, pending later resolution, or *terminated* which immediately aborts the transaction. See section 3.12 for details on fault behavior.

Events are recorded into the Event queue in response to a configuration error or translation-related fault associated with an incoming transaction. A sequence of faults or errors caused by incoming transactions could fill the Event queue and cause it to overflow if the events are not consumed fast enough. Events resulting from stalled faulting transactions are never discarded if the Event queue is full, but are recorded when entries are consumed from the Event queue and space next becomes available. Other types of events are discarded if the Event queue is full.

Note: ARM expects that the classes of events that might be discarded are generally used for debug.

Section 7.4 covers the exact queue behavior upon overflow.

ARM recommends that system software consumes entries from the Event queue in a timely manner to avoid overflow during normal operation.

In all cases in this document, when it is stated that an event is recorded, the meaning is that the event is recorded if room is available for a new entry in the Event queue and the queue is writable. A queue is writable if it is enabled, has no global error flagged and would not otherwise overflow, see section 7.2. Events that are not reported in response to a stalled transaction (for example where there is no Stall field, or Stall == 0) are permitted to be discarded if they cannot be recorded. Stall events are generally not discarded and are recorded when the

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Event queue is next writable, see section 7.2 for details of exceptions to this rule. Software must consume events from the queue to free up space, otherwise the pending stall events will not be recorded. Stall events are otherwise no different to any other event. The queue is filled in the same circular order and such events do not overwrite existing, unconsumed, events.

Where multiple pending events contend for a write to the Event queue, ARM recommends that an implementation does not unfairly prioritize non-stall events above events with Stall == 1, if it is possible to do so. This helps avoid the case of a steady stream of terminated transactions from a misbehaving device holding back the events of stalled transactions for an indeterminate time.

If an event is generated in response to a transaction that is terminated, there is no requirement for the event to be made visible in the Event queue before a transaction response is returned to the client. See CMD_SYNC, section 4.6.3, which enforces visibility of events relating to terminated transactions.

Note: This means that an event generated in response to a terminated transaction might be visible as an SMMU event before the point that the transaction termination is reported at the client device.

3.5.4 Definition of event record write "Commit"

Generation of an event record can be abstracted into these steps:

1. A situation that triggers an event occurs, for example a translation fault.
2. An event record is assembled internally in the SMMU.
3. It is determined that it is possible to write a new queue entry.
4. The final event record is committed to be written to the Event queue entry.
5. The event record becomes visible in the Event queue:
    a. The update to the record data location is visible to the required Shareability domain.
    b. The PROD.WR index is updated to publish the new record to software. In terms of queue semantics, the record is not visible (even if it has been written to memory) until the write index is updated to cover the new entry.

The commit point, 4, represents the conceptual point after which the event will definitely be written to the queue and eventually become visible. Until commit, the event write might not happen (for example, if the queue is full and software never consumes any entries, the event write will never commit).

An event write that has committed is guaranteed to become visible in the Event queue, if the subsequent write does not experience an external abort, see section 7.2.

The write of a stall event record must not commit until the queue entry is deemed writable (the queue is enabled and not full). If it is not writable, the stall record is buffered until the queue is next writable, unless one of the exceptions in section 7.2 causes the record to be discarded.

3.5.5 Event merging

Implementations are permitted to merge some event records together. This might happen where multiple identical events occurred, and can be used to reduce the volume of events recorded into the Event queue where individual events do not supply additional useful information.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Events can be merged where all of the following conditions are upheld:

- The event types and all fields are identical, except fields explicitly indicated in section 7.3.
- If present, the Stall field is 0. Stall fault records are not merged, see section 3.12.2.

An implementation is not required to merge any events, but one that does is required to support the STE.MEV flag to enable or inhibit merging of events relating to a given stream.

Note: For debugging purposes, merging of some events can be disabled on a per-stream basis using the STE.MEV flag.

Software implementations (for example a virtual emulation of an SMMU) are not required to respect STE.MEV. A hypervisor might cause events to continue to be merged after a guest requests merging to be disabled, for example if it determines a misbehaving guest to be causing too many debug events.

See section 7.3.1 for details.

3.6 Structure and queue ownership

ARM expects that the Non-secure Stream table, Command queue, Event queue and PRI queue are controlled by the most privileged Non-secure system software.

If present, ARM expects that the Secure Stream table, Secure Command queue and Secure Event queue are controlled by Secure software. For example, these would be controlled by software in EL3 if a separation in control between Secure EL1 and EL3 is required.

ARM expects that the stage 2 translation tables indicated by all STEs are controlled by a hypervisor.

The ownership of stage 1 CDs and translation tables depends on the configuration in use. If pointed to by a Secure STE, they are controlled by Secure software (either EL3 or EL1). If pointed to by a Non-secure STE, the context might be one of the following:

- Used by a bare-metal OS, which controls the descriptor and translation tables and is addressed by PA.
- Used internally by a hypervisor, which controls the descriptor and translation tables and is addressed by PA.
- Used by a guest, in which case ARM expects that the CD and translation tables are controlled by the guest, and addressed by IPA.

Note: ARM expects that the Non-secure Event queue is managed by the hypervisor, which forwards events into guest VMs as appropriate. StreamIDs might be mapped from physical to virtual equivalents during this process.

In virtualized scenarios, ARM expects a hypervisor to:

- Convert guest STEs into physical SMMU STEs, controlling permissions and features as required.

  Note: The physical StreamIDs might be hidden from the guest, which would be given virtual StreamIDs, so a mapping between virtual and physical StreamIDs must be maintained.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Read and interpret commands from the guest Command queue. These might result in commands being issued to the SMMU or invalidation of internal shadowed data structures.
- Consume new entries from the PRI and Event queues, mapping from host StreamIDs to guest, and deliver appropriate entries to guest Event and PRI queues.

3.7 Programming registers

The SMMU registers occupy a set of contiguous 64K pages of system address space that contain mechanisms for discovering capabilities and configuring pointers to in-memory structures and queues. After initialization, runtime access to the registers is generally limited to maintenance of the Command, Event and PRI queue pointers and interaction with the SMMU is performed using these in-memory queues.

Optional regions of IMPLEMENTATION DEFINED register space are supported in the memory map.

See section 6.1 for register definitions and layout.

3.8 Virtualization

Devices can be put under direct guest control with stage 2-only mappings, without requiring guest interaction with the SMMU. To the guest OS, they appear as though the device is directly connected and might request DMA to PAs (IPAs) directly.

The SMMU does not provide programming interfaces for use directly by virtual machines. ARM expects that, where stage 1 facilities are required for use by a guest in virtualization scenarios, this is supported using hypervisor emulation of a virtual SMMU, or a similar interface for use by a virtual machine.

Implementations might provide an IMPLEMENTATION DEFINED number of extra hardware interfaces that are located in an IMPLEMENTATION DEFINED manner but are otherwise compatible with the SMMUv3 programming interface. Each interface might be mapped through to a guest VM for it to use directly, for example appearing as a stage 1-only interface to the guest while the hypervisor interface appears as stage 2-only. The management of such an implementation is beyond the scope of this document.

3.9 Support for PCI Express, PASIDs, PRI and ATS

A PCI RequesterID maps directly to the low-order bits of a StreamID, therefore maps to one STE, see section 3.2. A PCIe Function is then the minimum granularity that can be assigned to a VM. The PCIe PASID prefix allows a Function to be subdivided into parts, each of which is intended to be assigned to a different user space process at stage 1. The prefix is optional. Transactions from one StreamID might be supplied with a SubstreamID or not, on a per-transaction basis. Because the prefix just identifies a portion of a Function, the Function remains otherwise indivisible and remains the granularity at which assignments to VMs are made. Therefore, in PCI terms:

- Stage 2 is associated with a RequesterID (identifying a Function). The Function is assigned to a VM.
- Stage 1 is associated with a (RequesterID, PASID) tuple. That is, the PASID differentiates between different stage 1 translation contexts.
- The PASID identifies which of the parts or contexts of the Function are assigned to which process or driver at stage 1.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- If transactions from a Function are translated using stage 2 but stage 1 is unused and in bypass, there are no stage 1 translation contexts to differentiate with a PASID. Supply of a PASID or SubstreamID to a configuration without stage 1 translation causes the translation to fail. Such transactions are terminated with an abort and C_BAD_SUBSTREAMID is recorded.

PASIDs can be up to 20 bits in size. PASIDs are optional, configurable, and of a size determined by the minimum of the endpoint, system software, PCIe Root Complex and the individually-supported substream width of the SMMU.

The SMMU is not required to report an error in the case where an endpoint and Root Complex emit a PASID with a value greater than can be expressed in the SubstreamID width supported by the SMMU. In this scenario, the PASID might be truncated to the SubstreamID size on arrival at the SMMU.

In order to minimize PCIe-specific terms, a RequesterID is referred to using a StreamID (to which the RequesterID maps in a hardware-specific manner). In the SMMU, a PASID is referred to as a SubstreamID. Even when a client device supports SubstreamIDs, it is not mandatory to supply a SubstreamID with all transactions from that device. PCIe permits a PASID to be supplied, or not, on a per-transaction basis. Therefore, where a SubstreamID is input to the SMMU, a validity flag is also provided and this is asserted when a PASID is present.

The PASID tag provides additional permission attributes on top of the standard PCIe read/write attribute. The tag can express an Execute and Privileged state that correspond to the SMMU INST and PRIV attributes. A PCIe transaction without a PASID is considered Data, unprivileged. The mapping between PCIe and SMMU permissions is described in section 13.7.

3.9.1 ATS Interface

An optional extra hardware interface might be provided by an SMMU implementation to support PCIe ATS [3] and PRI. This interface conveys translation and paging requests and responses to and from the PCIe Root Complex, which bridges requests and responses into the PCIe domain.

Whether the SMMU implements ATS can be discovered from SMMU_IDR0.ATS. If ATS is implemented, whether the SMMU also implements PRI can be discovered from SMMU_IDR0.PRI. This support determines the behavior of SMMU-local configuration and commands but does not guarantee that the rest of the system, and all clients of an SMMU, also support ATS and PRI. The ATS and PRI capabilities of dependent PCIe Root Complexes and endpoints thereof are discovered through other means.

PCIe ATS has the following properties:

- Note: ATS aims to improve the performance of a system using an SMMU, known as a *Translation Agent* in PCIe terminology, by caching translations within the endpoint or requester. This can remove contention on a shared TLB, or reduce latency by helping the device to request translation ahead of time.
- The remote endpoint Address Translation Cache (ATC, which is equivalent to a TLB) is filled on-demand by making a Translation Request to the Root Complex which forwards it to the SMMU. If the translation exists and permission checks pass, a Translation Completion response is given with a Physical Address and the ATC caches this response.
- The return of a translated, physical address to an endpoint grants the endpoint permission to access the physical address. The endpoint can now make direct access to PAs, which it does by tagging outgoing traffic as Translated. The Root Complex is expected to provide this tag to the SMMU. The SMMU might then allow such transactions to bypass translation and progress directly into the system.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- If a Translation Request would result in a fault or error, a negative response is returned and the endpoint is not able to access the address using ATS. This denial might be fatal to the endpoint, reported in a device-specific manner, or when PRI is used, initiate a page-in request.
- Page access permission checking is performed at the time of the Translation Request, and takes the form of a request for permission to read or to read/write (and, optionally, to execute). The response grants the device access to read or to read/write (and, optionally, to execute) as specified in [3]. The response returns the permissions that are available, which might consist of a subset of the requested permissions.

  Note: For example, a request to read and write a page might succeed, but only permission to read might be granted. The endpoint does not write pages it has not been granted write access to.
- ATS translation failures are reported to the endpoint, which might make an error software-visible, but the SMMU does not record fault events for ATS translation failures.
- Invalidation of the ATC translations is required whenever a translation changes in the SMMU. This is done in software. Broadcast invalidation operations might affect the internal translation caches of the SMMU, but these operations are not forwarded into the PCIe domain.
  - Note: An ARM broadcast TLB invalidation provides an address, ASID and VMID. The SMMU does not map this information back to the RequesterID of an endpoint in hardware.
  - Note: When CD.TBI0 or CD.TBI1 are used to enable use of tagged pointers with an endpoint that uses ATS, system software must assume that a given virtual address has been cached by the endpoint's ATC with any value of address bits [63:56]. This means that invalidation of a given virtual address VA[55:12] requires either 256 ATC invalidation operations to invalidate all possible aliases that the ATC might have cached, or an ATC invalidate-all operation.
- ATC invalidation is performed using SMMU commands which the SMMU forwards to the Root Complex. The invalidation responses are collected, and the SMMU maintains the ordering semantics upheld by the Root Complex in which a transaction that might be affected by an ATC invalidate must be visible before the ATC invalidation completes.
- ATS must be disabled at all endpoints before SMMU translation is disabled by clearing SMMU_CR0.SMMUEN.
- An ATS Translation Request might be fulfilled using SMMU TLB entries, or cause SMMU TLB entries to be inserted. Therefore, after a change of translation configuration, an ATC invalidate must be preceded by SMMU TLB invalidation. Software must ensure that the SMMU TLB invalidation is complete before initiating the ATC invalidation.

  Note: This order ensures that an ATS Translation Request performed after an ATC invalidate cannot observe stale cached translations.
- ATS and PRI are not supported from Secure streams.
  - In Secure STEs, the EATS field is RES0.
  - CMD_ATC_INV and CMD_PRI_RESP are not able to target Secure StreamIDs.
  - The SMMU terminates any incoming traffic marked Translated on a Secure StreamID, aborting the transaction and recording F_TRANSL_FORBIDDEN.
  - If it is possible for an implementation to receive an ATS Translation Request from a Secure StreamID, the request is aborted with a UR response and F_BAD_ATS_TREQ is recorded into the Secure Event queue. The check for a secure ATS Translation Request takes place prior to checking of StreamID or configuration lookup.
- Support for CMD_ATC_INV and CMD_PRI_RESP on the Secure Command queue is optional and is indicated by SMMU_S_IDR3.SAMS.
- The Smallest Translation Unit, (STU, as programmed into the ATS Control Register of the Endpoint) is defined as the smallest granule that the SMMU implements, as discovered from SMMU_IDR5. Software must program the same STU size for all devices serviced by an SMMU, and must not assume all SMMUs in the system are identical in this respect.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The Page Request Interface (PRI) adds the ability for PCIe functions to target DMA at unpinned dynamically-paged memory. PRI depends on ATS, but ATS does not mandate PRI. Like ATS, PRI can be enabled on a per-function basis. When enabled:

- If an ATS request fails with a not-present result, the endpoint issues a PRI page request to ask software to make the requested pages resident.
- Software receives these PRI Page Requests (PPRs) on the PRI queue and issues a positive PRI response command to the SMMU after making pages present. If a requested address is unavailable, a programming failure has caused the device to request an illegal address, and software must issue a negative PRI response command.
- The net effect is that a hypervisor or OS can use unpinned, dynamically-paged memory for DMA.
- The PRI queue is of a fixed size and PPRs must not be lost. To ensure this, page request credits are issued by the most privileged system software (that which controls the PRI queue) to each PCIe endpoint using the PRI capability in its configuration space.
- If a guest is allowed to use PRI, it enables PRI (through the configuration space) and sets up its own PRI queue. The hypervisor needs to proxy PPRs from the host PRI queue to the guest PRI queue. However, the total system number of PRI queue entries is limited by the PRI queue size of the hypervisor.
- The PRI queue size is limited up to a per-SMMU maximum, indicated by SMMU_IDR1.PRIQS. ARM expects that where PRI is used with virtualization then each guest discovers how many PRI queue entries its emulated SMMU supports. The host allocates N from its allocation of L, and ensures that the guest gives out a maximum of N credits (using configuration space) to devices controlled by the guest. L is the total number of PRI queue entries in the PRI queue of the host and is the maximum number of credits actually given out to devices.
- If the PRI queue becomes full because of erroneous behavior in a client device, the SMMU and Root Complex will respond to further incoming Page Request messages by returning a *successful* PRG response. This will not fatally terminate device traffic and a device will simply try ATS, fail, and try PRI again. ARM expects that a system employing this technique would remain functional and free-flowing, if requests were consumed from the PRI queue and space for new requests created, see section 8.1.

Note: ATS operation enables an endpoint to issue Translated requests that bypass the SMMU in some configurations. In these cases use of ATS could be a security issue, particularly when considering untrusted, subverted or non-compliant ATS devices. For example, a custom FPGA-based device might mark requests as Translated despite the ATS protocol and translation tables not having granted access.

SMMU_CR0.ATSCHK controls whether the SMMU allows Translated traffic to bypass with no further checks. If configured as requiring further checks, Translated requests from an endpoint are controlled by the associated STE.EATS field, which provides a per-device control of whether ATS traffic is allowed. When allowed, Translated requests are accepted, otherwise, an abort is reported and a F_TRANSL_FORBIDDEN event recorded.

Note: An implementation might perform this traffic interception and checking in a manner that is much quicker than performing full translation, thereby retaining a performance advantage of using ATS while achieving greater safety than permitting all ATS traffic.

STE.EATS also allows a mode in which ATS responses are returned with IPAs, the output from the stage 1 of a stage 1 and stage 2 configuration, so that later Translated requests from the endpoint are considered IPAs and further translated by the SMMU using the stage 2 configuration of the stream. This allows ATS to be used (for example with PRI) while maintaining stage 2 isolation. This mode is optional in an implementation, and support is discovered through SMMU_IDR0.NS1ATS. When implemented, this mode can only be used when SMMU_CR0.ATSCHK forces traffic not to bypass the SMMU, as stage 2 translation needs to be applied.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: When ATS is used with nested stage 1 and stage 2 translation, any modification to stage 1 or stage 2 requires an invalidation of ATC entries, which cache information derived from both stages. This also applies to the EATS == `0b10` Split-stage ATS case.

When Translated transactions bypass the SMMU, an incoming address greater than the output address size (OAS) has one of the following CONSTRAINED UNPREDICTABLE behaviors:

- The incoming address is truncated to the OAS. The SMMU does not provide address size error-checking on Translated transactions.
- The transaction is terminated with an abort. No event is recorded.

Note: This situation would not normally arise outside of incorrect ATC invalidation when transitioning between Split-stage ATS mode and regular ATS mode.

ARM expects that a typical implementation connects an SMMU to a PCIe Root Complex so that incoming StreamIDs are generated directly from PCIe RequesterIDs. The ATS Interface requires a simple transformation between RequesterID and StreamID, so that StreamID[15:0] == RequesterID[15:0]. Where multiple Root Complexes are connected to one SMMU, the upper bits of StreamID, StreamID[N:16], correspond to the Root Complex supporting a given endpoint.

ARM expects that most highly-integrated non-PCIe devices requiring translation and paging facilities will use implementation-specific distributed SMMU TLB facilities, rather than using ATS and PRI. Using SMMU facilities allows such devices to participate in broadcast TLB invalidation and use the Stall fault model.

If the translation requested by an ATS request is valid and HTTU is enabled, the SMMU must update Translation Table Dirty/Access flags on receipt of the ATS translation request, see 3.13 below An ATS request is either for a read-only translation (in which case the NW flag of the request is set) so only the Access flag is updated, or for a read-write translation (in which case the NW flag of the request is clear) for which both the Access flag and the Dirty state of the page are updated.

Note: Because the intention is for the actual traffic to bypass the SMMU, the ATS request is the only opportunity the SMMU will have to note the access in the flags.

A PASID tag can also be applied to an ATS Translation Request to select translation under a specific SubstreamID. A PASID-tagged ATS TR requests that the endpoint be granted access to a given address, according to the Execute and Privileged attributes of the PASID in addition to the existing NW write intention.

When the SMMU returns an ATS Translation Completion for a request that had a PASID, the Global bit of the Translation Completion Data Entry must be zero.

Note: The SMMU differentiates translation contexts intended to be shared with the PE from those not shared, using the CD.ASET mechanism. Whether a global translation matches is also a function of ASET. However, no mechanism exists to indicate that all possible global translations (from all contexts used by an endpoint) share an identical address space layout so that global translations can be used. The ATS Global flag must be cleared because a non-shared context must not match global translations from a shared context (and vice versa).

Note: ARM expects that general-purpose software will require HTTU for use with PRI. See section 3.13.6 for more information on flag updates with ATS.

Note: PRI requires ATS to be implemented, but ATS does not require PRI to be implemented.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: An SMMU that does not support HTTU can support paged DMA mappings for non-PCI devices using the Stall fault model, see section 3.12. PCIe cannot be used with the Stall fault model, so a requirement for paged DMA with PCIe implies a requirement for PRI, which implies a requirement for HTTU.

Transactions that make use of ATS might differ from ordinary PCIe non-ATS transactions in several ways:

- Translation Requests that do not successfully translate, including those that would ordinarily have CD.A == 0 RAZ/WI behavior, cause an error in the endpoint (recorded in an endpoint-defined manner) or a PRI request, instead of an error or fault being recorded using the SMMU Event queue.
- Changes to translations require use of CMD_ATC_INV in addition to SMMU TLB invalidation.
- ATS Translated transactions might not represent Instruction/Data and/or Privileged/User marking on the interconnect to memory in the same way as Untranslated transactions.
- Pages with execute-only and no read, writable executable permissions cannot be represented, and are inaccessible when using ATS, see section 13.7.

3.9.1.1 Responses to ATS Translation Requests and Translated transactions

A Translation Request made from a StreamID for which ATS is explicitly or implicitly disabled (because of SMMU_CR0.SMMUEN == 0, or the effective EATS == `0b00` including where this is because of a Secure STE, or STE.Config == `0b000`) results in an ATS Translation Completion with Unsupported Request (UR) status.

Configuration or scenario	For an ATS Translation Request, leads to
SMMUEN == 0	Terminated with UR status and F_BAD_ATS_TREQ generated
Using a Secure StreamID	Terminated with UR status and F_BAD_ATS_TREQ generated
STE.Config == 0b000	Terminated with UR status
STE.Config == 0b100	Terminated with UR status and F_BAD_ATS_TREQ generated
Effective STE.EATS == `0b00` (Note: Includes EATS == `0b10` when ATSCHK == 0)	Terminated with UR status and F_BAD_ATS_TREQ is generated

A Translation Request that encounters an Address Size, Access or Translation fault arising from the translation process for a page, at either stage, results in an ATS Translation Completion with Success status and R == W == 0 in the Completion Data Entry for that page and no fault is recorded in the SMMU. If the R == W == 0 Translation Completion Data Entry is the first or only entry in the Translation Completion, its translation size is equal to the STU size. A Permission fault can also lead to this response, but other cases that would cause a Permission fault for an ordinary transaction might result in some, but not all, permissions being granted to the endpoint. See 13.7 for information on permission calculation for ATS.

Consistent with ARMv8-A, in a two-stage translation, the IPA to PA translation of the output address of a stage 1 table, block, or page descriptor is not architecturally performed unless the descriptor is valid and no fault would arise from the descriptor. This behavior applies to a two-stage translation that is performed for an ATS Translation Request, which means that translation stops if stage 1 leads to an Address Size, Access, or Translation fault, or evaluates to a Translation Completion that grants no permissions. If the final IPA for stage 1 is valid, but does not provide any access permissions, the IPA is not translated at stage 2, and no error responses from stage 2 are visible.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: The case of Split-stage ATS is included, because permissions are determined from both stages. For more information, see section 13.6.3.

A Translation Request that encounters any configuration error (for example ILLEGAL structure contents, or external abort on structure fetch) results in an ATS Translation Completion with Completer Abort (CA) status:

If an ordinary transaction were to trigger a…	…an ATS Translation Request with the same properties leads to:
C_BAD_STREAMID F_STE_FETCH C_BAD_STE F_CFG_CONFLICT F_TLB_CONFLICT C_BAD_SUBSTREAMID F_STREAM_DISABLED F_WALK_EABT F_CD_FETCH C_BAD_CD	Terminate with CA status
F_ADDR_SIZE F_ACCESS F_TRANSLATION	Success: R == W == 0 (access denied) This includes stage 2 faults for a CD fetch or stage 1 translation table walk.
F_PERMISSION	Success. R, W and Exe permission is granted, where requested, from available translation table permissions. In the extreme case, a translation with no access permission gives R == W == 0. Where F_PERMISSION arises at stage 2 for a CD fetch or stage 1 translation table walk, the response of Success and R == W == 0 is given.

Translated transactions generally pass through the SMMU unless the Non-secure SMMUEN is disabled, a Secure stream is used, or if ATSCHK == 1 and therefore additional configuration checks are performed.

Configuration/scenario	For a Translated Transaction, leads to:	
SMMUEN == 0	F_TRANSL_FORBIDDEN and aborted	
Using a Secure StreamID	F_TRANSL_FORBIDDEN and aborted	
STE.Config == `0b000`	If ATSCHK == 1	Aborted
STE.Config == `0b100`		F_TRANSL_FORBIDDEN and aborted
Effective STE.EATS == `0b00`		F_TRANSL_FORBIDDEN and aborted

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If an ordinary transaction were to trigger a…	…a Translated transaction with the same properties is:
F_UUT	Aborted
C_BAD_STREAMID F_STE_FETCH C_BAD_STE F_CFG_CONFLICT	If ATSCHK == 1, aborted. Note: If ATSCHK == 0, the SMMU does not check configuration for Translated transactions, so does not detect these conditions.

If a Translated transaction experiences a second stage 2 translation because of an STE.EATS == `0b10` configuration, the transaction is terminated with an abort the same way as any ordinary transaction when a fault occurs during stage 2 translation (F_WALK_EABT, F_TLB_CONFLICT, F_ADDR_SIZE, F_ACCESS, F_TRANSLATION, F_PERMISSION).

Note: Since a Translated transaction does not have a PASID, it is presented to the SMMU with PnU == 0, InD == 0 and SSV == 0. A resulting fault is reported as Stage 2, with the input address as given by the Translated transaction.

If STE.S1DSS causes a stage 1 skip, and STE.Config == `0b101` (stage 1-only), the response is Success, U == 0, R == W == 1, identity-mapping (see 13.6).

3.9.1.2 ATC invalidation timeout

A CMD_ATC_INV causes an ATS Invalidate Request to be sent to an endpoint and, in the case that a response is not received within the timeout period specified by ATS, ARM strongly recommends the following behavior:

- The Root Complex isolates the endpoint in a PCI-specific manner, if it is possible to do so.
- A CMD_SYNC that waits for completion of one or more prior CMD_ATC_INV operations causes a CERROR_ATC_INV_SYNC command error if any of the CMD_ATC_INV operations have not successfully completed. See 7.1.
    - Note: Command processing stops and this situation is differentiated from a normal completion of a CMD_SYNC, which avoids the potential re-use and corruption of a page that has been unmapped but whose translation was incorrectly invalidated.
- If it is not possible for an implementation to cause CERROR_ATC_INV_SYNC for a CMD_SYNC that waits for the completion of failed CMD_ATC_INV operations, ARM recommends that the CMD_SYNC does not complete.
    - Note: This scenario is not recoverable but prevents the invalidation from appearing to have completed, leading to potential data corruption (the error is contained and propagation is avoided).
- In the event of a failed CMD_ATC_INV, ARM strongly recommends that a related CMD_SYNC does not complete without raising a command error. An IMPLEMENTATION DEFINED error mechanism asynchronous to the completion of the CMD_SYNC must record information of the failure.
    - Note: A completion of a CMD_SYNC without completing an invalidation might lead to corruption of a page that is subsequently re-used by different mappings.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.9.1.3 ATC invalidation errors

A CMD_ATC_INV that generates an ATS Invalidate Request that causes a UR response from an endpoint completes without error in the SMMU. An invalidation might not have been performed in response to the command.

Note: A UR response to an invalidation can occur in several circumstances as specified by [3], including where an invalidation is sent with an out-of-range PASID value.

3.9.2 Changing ATS configuration

The ATS behavior of an endpoint is dependent on the STE.EATS field that is associated with the endpoint and on SMMU_CR0.ATSCHK. In addition to enabling extra checks on Translated transactions, ATSCHK changes the interpretation of the EATS == `0b10` encoding, and because ATSCHK is permitted to be cached in configuration caches, this means that a change to ATSCHK must be followed by invalidation of any STEs that are required to heed the new value.

Note: The EATS encodings of `0b01` and `0b10` will respond to Translation Requests and interpret Translated transactions using different address spaces. A direct transition between these encodings might cause IPAs to be interpreted as PAs or vice-versa, which might lead to data corruption.

To enable ATS on an existing valid STE with EATS == `0b00`:

1. EATS is set to `0b01` or `0b10` and caches of the STE are invalidated (including CMD_SYNC to ensure completion of the invalidation)
2. ATS is enabled at the endpoint.

To disable ATS on an existing STE with EATS != `0b00`:

1. ATS must be disabled at the endpoint, the ATCs invalidated, and CMD_SYNC used to ensure visibility of prior transactions using ATS that are in progress.
2. EATS is then set to `0b00`.
3. Caches of the STE are then invalidated.

EATS must not be transitioned between `0b01` and `0b10` (in either direction) without first disabling ATS with the procedure described in this section, transitioning through EATS == `0b00`.

EATS == `0b10` is valid only when SMMU_CR0.ATSCHK == `1`. ATSCHK must not be cleared while STE configurations (and the possibility of caches thereof) exist with EATS == `0b10`. Before clearing ATSCHK, all STE configurations with EATS == `0b10` must be re-configured to use EATS == `0b00` or EATS == `0b01`, using the procedures described in this section.

Note: This ensures that Translated traffic using IPA addressing (originating from Translation Requests handled by a stage 1-only EATS == `0b10` configuration) does not encounter an SMMU with ATSCHK == 0, which would pass the traffic into the system with a PA.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Although ATSCHK == 0 causes EATS == `0b10` to be interpreted as `0b00` (ATS disabled), ATSCHK must not be used as a global ATS disable.

To set ATSCHK to 1:

1. Set SMMU_CR0.ATSCHK == 1 and wait for Update procedure to complete.
2. STEs (pre)fetched after this point will interpret STE.EATS according to the new ATSCHK value.
3. Unexpected Translated traffic that is associated with an STE with EATS == `0b00` will now be terminated.
4. ATS can be enabled on an STE as described here:
    a. Note: The STE update procedure invalidates the STE, which will invalidate any old ATSCHK value cached with it.

To clear ATSCHK to 0:

1. Ensure that the ATS is disabled for all STEs that were using EATS == `0b10`, flushing ATCs and transitioning through EATS == `0b00`.
    a. Note: After this point, there will be no relevant caches of ATSCHK.
2. Set SMMU_CR0.ATSCHK == 0 and wait for the Update procedure to complete.
3. STEs (pre)fetched after this point will interpret STE.EATS according to the new ATSCHK value.
4. Translated traffic now bypasses the SMMU without additional checks.
5. Split-stage ATS cannot be enabled on an STE, meaning EATS == `0b10` must not be used.

Referring to section 13.6.3 and 13.6.4, it is possible to configure ATS for a stream where only requests made from substreams (PASIDs) return actual translations, and non-substream Translation Requests return an identity-mapped response that might be cached at the endpoint. Substream configuration (STE.S1DSS and STE.S1CDMax) therefore affects the contents of ATS Translation Completion responses and any change of this configuration must also invalidate endpoint ATCs.

3.10 Support for two Security states

The ARM architecture provides support for two Security states, each with an associated physical memory address space:

- Secure state (NS == 0).
- Non-secure state (NS == 1).

SMMU_S_IDR1.SECURE_IMPL indicates whether an SMMU has been configured to support the ARM Security model.

When SMMU_S_IDR1.SECURE_IMPL == 0:

- The SMMU supports only a single Security state, the Non-secure state.
- SMMU_S_* registers are RAZ/WI to all accesses.
- The SMMU can only generate Non-secure (NS == 1) transactions to the memory system.
- Support for stage 1 translation is OPTIONAL.

When SMMU_S_IDR1.SECURE_IMPL == 1:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The SMMU supports two Security states, Secure state and Non-secure state.
- SMMU_S_* registers configure a Secure Command queue, Secure Event queue and a Secure Stream table.
- The SMMU supports stage 1 translation.
- The SMMU can generate both Secure (NS == 0) and Non-secure (NS == 1) transactions to the memory system, where permitted by SMMU configuration.

With the exception of SMMU_S_INIT, SMMU_S_* registers are Secure access only, and RAZ/WI to Non-secure accesses.

Note: ARM does not expect a single software driver to be responsible for programming both the Secure and Non-secure interface. However, the two programming interfaces are intentionally similar.

When a stream is identified as being under Secure control by the Secure State Determination process, see 3.10.1 below, its configuration is taken from the Secure Stream table or from the global bypass attributes that are determined by SMMU_S_GBPA. Otherwise, its configuration is taken from the Non-secure Stream table or from the global bypass attributes that are determined by SMMU_GBPA. The Secure programming interface and Non-secure programming interface have separate global SMMUEN translation-enable controls that determine whether bypass occurs. See section 3.10.1 below for more information about the Secure State Determination process.

A transaction that belongs to a Stream that is under Secure control can generate Secure (NS == 0) and Non-secure (NS == 1) transactions to the memory system. A transaction that belongs to a Stream that is under Non-secure control can only generate Non-secure (NS == 1) transactions to the memory system.

3.10.1 Secure State Determination (SSD)

Secure State Determination (SSD) determines whether a given transaction stream is under the control of the Secure or the Non-secure programming interface.

The association between a device and the Secure or Non-secure programming interface is a system-defined property.

When SMMU_S_IDR1.SECURE_IMPL == 1, incoming transactions have a StreamID and a SEC_SID flag:

- When SEC_SID == 1, the stream is a Secure stream, and indexes into the Secure Stream table.
- When SEC_SID == 0, the stream is a Non-secure stream, and indexes into the Non-secure Stream table.

The system contains two StreamID namespaces, one for Non-secure StreamIDs and one for Secure StreamIDs. As indicated in this section, the assignment of a client device to either a Secure StreamID or a Non-secure StreamID, and moving StreamID namespaces, is system-defined.

In this document, the terms Secure StreamID and Secure stream refer to a stream that is associated with the Secure programming interface, as determined by the SSD. The terms Non-secure StreamID and Non-secure stream refer to a stream that is associated with the Non-secure programming interface, as determined by the SSD.

Note: Whether a stream is under Secure control or not is a different property to the NS attribute of a transaction if a stream is Secure, it means that it is controlled by Secure software through the Secure Stream table. Whether a

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

transaction on that stream results in a transaction with NS == 0 depends on the translation table attributes of the configured translation, or, for bypass, the incoming NS attribute.

3.10.2 Secure commands, events and configuration

In this document, the term Event queue and the term Command queue refer to the queue that is appropriate to the Security state of the relevant stream. Similarly, the term Stream table and Stream table entry (STE) refer to the table or table entry that is appropriate to the Security state of the stream as indicated by SSD.

For instance:

- An event that originates from a Secure StreamID is written to the Secure Event queue.
- An event that originates from a Non-secure StreamID is written to the Non-secure Event queue.
- Commands that are issued on the Non-secure Command queue only affect streams that are configured as Non-secure.
- Some commands that are issued on the Secure Command queue can affect any stream or data in the system.
- The stream configuration for a Non-secure StreamID X is taken from the Xth entry in the Non-secure Stream table.
- Stream configuration for a Secure StreamID Y is taken from the Yth entry in the Secure Stream table.

The Non-secure programming interface of an SMMU with SMMU_S_IDR1.SECURE_IMPL == 1 is identical to the interface of an SMMU with SMMU_S_IDR1.SECURE_IMPL == 0.

Note: To simplify descriptions of commands and programming, this document refers to the Non-secure programming interface registers, Stream table, Command queue and Event queue even when SMMU_S_IDR1.SECURE_IMPL == 0.

The register names associated with the Non-secure programming interface are of the form *SMMU_x*. The register names associated with the Secure programming interface are of the form *SMMU_S_x*. In this document, where reference is made to a register but the description applies equally to the Secure or Non-secure version, the register name is given as *SMMU_(S_)x*. Where an association exists between multiple Non-secure, or multiple Secure registers and reference is made using the *SMMU_(S_)x* syntax, the registers all relate to the same Security state unless otherwise specified.

The two programming interfaces operate independently as though two logical and separate SMMUs are present, with the exception that some commands issued on the Secure Command queue and some Secure registers might affect Non-secure state, as indicated in this document. This independence means that:

- The Command and Event queues that are associated with a programming interface operate independently of the Command and Event queues that are associated with the other programming interface. The operation of one does not affect the other programming interface, for example when:
    - The queues are full.
    - The queues overflow.
    - The queues experience an error condition, for example a Command queue that stops processing because of a command error, or an abort on queue access.
- Translation through each programming interface can be separately enabled and disabled using the SMMUEN field that is associated with the particular programming interface. This means that one

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

interface might bypass transactions in which case the behavior is governed by the respective SMMU_(S_)GBPA and the other programming interface might translate transactions.

- Error conditions in SMMU_(S_)GERROR apply only to the programming interface with which the register is associated.
- Each interface has its own ATOS interface, where ATOS is implemented.
- Interrupts are configured and enabled separately for the Secure and Non-secure programming interface interrupt sources.

When SMMU_S_IDR1.SECURE_IMPL == 1, ARM expects that the SMMU will be hosted by a PE implementing two Security states. The host PE might:

- Implement ARMv7-A.
- Implement ARMv8-A, with EL3 using AArch64 state.
- Implement ARMv8-A, with EL3 using AArch32 state.

StreamWorld differentiates the Secure EL1 translation regime from the EL3 translation regime, allowing TLB entries to be maintained separately for each of these two translation regimes. Secure EL1 TLB entries might be tagged with an ASID, whereas EL3 TLB entries are not. In this case, ARM expects that the Secure SMMU interface is either:

- Managed by Secure EL1, with no SMMU usage by EL3.
- Managed by EL3 with any EL1 usage brokered to EL3 using a software interface, which is outside the scope of this document.

ARM recommends that Secure EL1 and EL3 do not attempt to both access the Secure Command queue. ARM further recommends that Secure EL1 does not configure streams to cause TLB entries to be marked as EL3.

When the SMMU is controlled by a PE that implements ARMv8-A and uses AArch32 in EL3 or a PE that implements ARMv7-A, there is only one privileged Secure translation regime. No separation is made between Secure OS and Secure monitor TLB entries. When used with this type of system, ARM recommends that the StreamWorld is configured so that TLB entries that are associated with this Secure translation regime are ASID-tagged. In this case, ARM recommends that StreamWorld is not configured to insert EL3 TLB entries, because broadcast TLB invalidation from the PE would not be able to affect these TLB entries. For more information, see section 3.17.

A client device that is SSD Secure provides an attribute called *NSPROT* that indicates whether an access is intended to be Secure or Non-secure. A Secure STE might override the NSPROT attribute of a stream.

In bypass configurations of a Secure stream, overriding the NSPROT attribute allows a client device to issue Secure accesses even if the device is not able to control its NSPROT attribute. If the NSPROT attribute is not overridden, the client device can control whether it makes Secure or Non-secure accesses.

When a Secure STE is configured for stage 1 translation, the stage 1 translation table descriptor (in conjunction with intermediate NSTable bits) determines the output NS attribute if the TTD is fetched from Secure memory, in the same way as in a PE and in the SMMUv2 architecture. See section 13.

A Non-secure STE does not override NSPROT, which is treated as 1 (Non-secure) for all transactions belonging to a Non-secure stream.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Access to the Secure Stream table, CDs and L1CDs located through Secure STEs, the Secure Event queue and the Secure Command queue are made with NS == 0.

Some SMMU commands take a StreamID parameter. When issued from the Secure Command queue, an additional parameter that is equivalent to SEC_SID indicates whether the command interprets the StreamID as Secure or Non-secure.

The SMMU_S_CR0.SIF flag provides a mechanism to terminate transactions from a Secure stream when they are marked as an instruction fetch and are assigned an NS attribute of Non-secure.

3.11 Reset, Enable and initialization

The SMMU can reset to a disabled state in which traffic bypasses the SMMU without translation or checking of any kind. The SMMU appears transparent to transactions from client devices, which are given attributes according to the disabled bypass configuration (see section 13). The SMMU can also optionally reset to a disabled state that aborts all transactions for a Security state. This behavior is controlled by the reset state of SMMU_GBPA.ABORT or SMMU_S_GBPA.ABORT.

Note: When an SMMU resets to a bypass configuration, it enables client devices that are connected to an SMMU to be used by legacy system software that lacks awareness of the SMMU.

Translation of Non-secure Streams is enabled using SMMU_CR0.SMMUEN. When SMMU_S_IDR1.SECURE_IMPL == 1, the Secure programming interface also contains an enable flag, SMMU_S_CR0.SMMUEN, which controls translation of Secure streams.

When translation is not enabled for a Security state, an SMMU:

- When SMMU_(S_)GBPA.ABORT == 1, aborts all transactions:
- When SMMU_(S_)GBPA.ABORT == 0, applies attributes to a transaction as determined by SMMU_(A)GBPA or SMMU_S_(A)GBPA. See section 13.2.
- Never accesses the Stream table so SMMU_(S_)STRTAB_* register content is ignored.
- Denies PRI Page Requests as though SMMU_CR0.PRIQEN == 0, regardless of the value of SMMU_CR0.PRIQEN. See section 8.
- Does not perform ATOS operations. See 6.3.37.
- Does not perform ATS translations. See 3.9.1.1.
- Allows registers to be accessed and updated in the normal manner.
- Can process commands after the relevant queue pointers are initialized and SMMU_(S_)CR0.CMDQEN is enabled.
- Does not record new translation events. However, if SMMU_(S_)CR0.EVENTQEN is enabled and the queue pointers are set up, the SMMU might continue to write out buffered events that were generated by earlier translations from when translation was still enabled.

See section 6.3.9.6 for a full description of the operation of, and the effect of changes to, the SMMUEN flag.

The SMMU_(S_)STRTAB_BASE register and the SMMU_(S_)CR1 table attributes must be configured before enabling an SMMU interface using SMMU_(S_)CR0.SMMUEN.

Note: This avoids the possibility of incoming traffic attempting a lookup through uninitialized configuration structure pointers.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When translation is disabled for a Security state, transactions on streams that are associated with that Security state are not translated, and take attributes from the appropriate Global Bypass Attribute registers, SMMU_(A)GBPA or SMMU_S_(A)GBPA.

When translation is enabled for a Security state, transactions on streams that are associated with that Security state follow the SMMU translation flow determined by the appropriate Stream table entry.

SMMU_CR0.SMMUEN	SMMU_S_CR0.SMMUEN	
0	Unimplemented	All traffic bypasses SMMU/aborts (as determined by SMMU_GBPA.ABORT). Always outputs NS == 1.
1	Unimplemented	Traffic follows the SMMU translation flow. Always outputs NS == 1.
0	0	Secure and Non-secure streams are controlled by SMMU_(S_)(A)GBPA. SSD determines the Security state of a given stream. Bypass/abort configuration and attributes, including NS, are provided by the Global Bypass Attribute register (GBPA) appropriate to the Security state.
0	1	SSD determines the Security state: • Secure traffic follows SMMU translation flow • Non-secure traffic bypasses the SMMU (attributes taken SMMU_GBPA), or aborts
1	0	SSD determines Security state: • Secure traffic bypasses the SMMU (attributes taken from SMMU_S_GBPA), or aborts • Non-secure traffic follows the SMMU translation flow
1	1	SSD determines the Security state, follows usual SMMU translation flow.

The state of the caches and TLBs at reset is implementation specific.

To avoid UNPREDICTABLE behavior, software must perform the following steps before enabling translation:

- Invalidate all configuration and TLB caches
- When SMMU_S_IDR1.SECURE_IMPL == 1, ensure Secure software fully invalidates any Secure cached configuration or TLB entries in the SMMU through the Secure programming interface before handover to Non-secure software.

The SMMU is not required to invalidate cached configuration or TLB entries when a change to SMMU_(S_)CR0.SMMUEN occurs.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

ARM recommends that software initializing the SMMU performs the following steps:

1. Allocate and initialize Stream table memory and base pointers.
2. Allocate and initialize Command queue and Event queue memory, base pointers and indexes.
3. Enable command processing through SMMU_(S_)CR0.CMDQEN, and if applicable, Event queue through the relevant EVENTQEN.
4. Issue commands to invalidate all cached configuration and TLB entries (see sections 4.3 and 4.4).
5. Enable translation by setting SMMU_(S_)CR0.SMMUEN.

Note: These steps are a summary, and do not show the required register update procedure or DSB operations ensuring correct memory and register access ordering.

SMMU_S_INIT invalidates SMMU caches and TLBs without issuing commands using the Command queue. Caches and TLBs are invalidated using this register with the following sequence:

- Perform a write to SMMU_S_INIT, setting INV_ALL.
- Poll SMMU_S_INIT.INV_ALL until it returns to 0, at which point the invalidation is complete.

When SMMU_S_IDR1.SECURE_IMPL == 1, ARM expects Secure software to initialize the SMMU using the steps above. If Secure software is not guaranteed to initialize the SMMU in accordance with the steps above, ARM recommends that the system provides an IMPLEMENTATION DEFINED mechanism to allow Non-secure software to access SMMU_S_INIT. This is an exception to the general rule that only Secure software can access SMMU_S_* registers.

Note: For example, a system might allow Non-secure access to SMMU_S_INIT from reset, but might provide a means for Secure software to disable this access.

Note: ARM expects Non-secure initialization to use SMMU commands to perform configuration cache and TLB invalidation. Non-secure access to SMMU_S_INIT is not guaranteed, so the INV_ALL feature must not be relied on by the Non-secure state.

If an SMMU implementation creates TLB entries when bypass is selected with SMMUEN == 0, these entries are not visible to software. An implementation does not require TLB entries inserted to support transaction bypass to be explicitly invalidated by software, such as when SMMUEN is transitioned from 0 to 1.

3.12 Fault models, recording and reporting

An incoming transaction goes through several logical stages before continuing into the system. If the transaction is of a type or has a property that an SMMU cannot support for IMPLEMENTATION DEFINED reasons, an Unsupported Upstream Transaction fault event is recorded and the transaction is terminated with an abort.

Otherwise, configuration is located for the transaction, given its StreamID (and SubstreamID, if supplied). If all of the required STE and CD structures cannot be located or are invalid, a configuration error event is recorded, if there is a free location in the Event queue, and the transaction is terminated with an abort.

If a valid configuration is located so that the translation tables can be accessed, the translation process begins, other faults can occur during this phase. See section 7.2 for more information about the individual events that are recorded for configuration errors and faults.

When a transaction progresses as far as translation, or during the process of fetching a CD from IPA space through stage 2 translation, the behavior on encountering a fault becomes configurable, if this is supported by the implementation.

There are four fault types that constitute Translation-related faults when they are generated at either stage 1 or stage 2:

- F_TRANSLATION
- F_ADDR_SIZE

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- F_ACCESS
- F_PERMISSION

Behavior for these faults can be switched between the Terminate and Stall model as determined by the CD.{A,R,S} flags for stage 1 and the STE.{S2R,S2S} flags for stage 2.

All other faults (including F_WALK_EABT and F_TLB_CONFLICT) and configuration errors terminate the transaction with an abort.

Note: An F_ADDR_SIZE can also arise from a transaction that bypassed stage 1 but that has an out-of-range IPA, see section 3.4. In this case the transaction is always terminated with an abort.

Note: An F_PERMISSION can also arise as a result of an instruction fetch transaction on a Secure stream that bypasses stage 1, is determined to be Non-secure and that is prevented with SMMU_S_CR0.SIF == 1, see section 6.3.47. In this case the transaction is always terminated with an abort.

The fault behavior configuration at stage 1 is at a per-substream granularity when substreams are used, that is where an STE points to multiple CDs. When substreams are not configured, that is where an STE points to one CD, the fault behavior configuration at stage 1 is at a per-stream granularity. Use of the Stall model at stage 1 can be disabled by setting STE.S1STALLD == 1.

The stage 2 fault behavior is configured using STE.{S2R,S2S}; that is, at a per-stream granularity.

When a fault occurs at either stage 1 or stage 2, then when the fault is detected it is known at which stage it occurred, and the SMMU performs the action configured for that stage. For example:

- A two-stage configuration that encounters a translation fault in the stage 1 translation tables is a stage 1 fault.
- A transaction that progresses through stage 1 to an IPA and then faults when it is translated using the stage 2 translation tables is a stage 2 fault.
- A stage 2 translation fault that occurs during a stage 1 translation table walk counts as a stage 2 fault. The event that is recorded differentiates this access from a transaction that access a faulting IPA post-stage 1 translation table walk, so that hypervisor software can inform the VM of the correct event type (a simulated external abort on TTD fetch).
- A Stage 2 translation fault that occurs fetching a CD from an IPA address is a stage 2 fault. The event that is recorded shows that a CD was being fetched.

  Note: The Hypervisor might fix the cause of the fault and retry the stalled transaction, or if the transaction is terminated, inform the VM of the correct event type (a simulated external abort on CD fetch).

After a transaction progresses through the SMMU into the system, certain system-specific transaction aborts might occur on the path to the memory system. Whether, and how, these are reported to the client device is interconnect-specific. The SMMU does not record any faults for these events. The SMMU only records fault events that are generated by its own accesses or by client device accesses that encounter an internal translation issue.

When an incoming transaction is immediately terminated, for any reason, an order is not enforced between the response to the client device and the event that is recorded into the Event queue. However, if an event is to be recorded, a CMD_SYNC ensures that the event record is visible in the Event before the CMD_SYNC is considered complete. See section 4.6.3.

The SMMU treats a transaction as being independent of all other transactions (regardless of whether the transaction originates from the same traffic stream or from different streams) and the fault behavior of one transaction has no direct effect on any other transaction. Section 3.12.2 below describes interconnect ordering

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

issues and recommendations for the presentation of grouped fault information to software. Whether an external agent makes an association between different transactions is outside the scope of the SMMU architecture.

When a transaction causes a Translation related fault at stage 1, the transaction might be:

- Terminated with an abort (CD.S == 0 and CD.A == 1)
- Terminated with RAZ/WI behavior (CD.S == 0 and CD.A == 0)
- Stalled (CD.S == 1 and STE.S1STALLD does not prevent stalling)

When a transaction Translation related fault at stage 2, the transaction might be:

- Terminated with an abort (STE.S2S == 0)
- Stalled (STE.S2S == 1)

Support for stalling or terminating a transaction is IMPLEMENTATION DEFINED, indicated by SMMU_(S_)IDR0.STALL_MODEL.

When SMMU_S_IDR1.SECURE_IMPL == 1:

- SMMU_S_IDR0.STALL_MODEL indicates the physical capabilities of the SMMU implementation,
- SMMU_IDR0.STALL_MODEL indicates the capabilities that Non-secure software is permitted to use.
    - This field is generated from SMMU_S_IDR0.STALL_MODEL and affected by the SMMU_S_CR0.NSSTALLD flag which, when set on an SMMU implementation supporting both the Stall model and the Terminate model, prevents Non-secure use of stalling faults.
    - Note: This can be used to guarantee Non-secure software cannot stall transactions where doing so might cause external problems in certain system topologies.

When SMMU_S_IDR1.SECURE_IMPL == 0, SMMU_IDR0.STALL_MODEL reflects the physical capabilities of the SMMU implementation.

SMMU_S_IDR0. STALL_MODEL	**SMMU_S_CR0. NSSTALLD**	**SMMU_IDR0. STALL_MODEL**	**Notes:**
`0b00` (Stall and Terminate models supported)	0 (do not filter NS use of stall)	`0b00` (Stall and Terminate model supported for NS)	NS usage reflects physical reality
`0b00` (Stall and Terminate models supported)	1 (NS cannot use stall)	`0b01` (Terminate model supported for NS)	NS usage limited to terminate-only, even though physically the SMMU supports stall too.
`0b01` (Terminate model supported)	X (No stall to filter)	`0b01` (Terminate model supported)	NSSTALLD irrelevant, no stall to prevent
`0b10` (Stall model supported)	X (No alternative to stall)	`0b10` (Stall model supported)	NSSTALLD irrelevant, no alternative to stall so cannot disable

The SMMU_IDR0.TERM_MODEL field indicates the termination models provided by an implementation, globally. An implementation might, for a stage 1 fault, offer the choice of terminate with abort or RAZ/WI behavior, or an implementation might only allow termination by abort, in which case the CD.A bit must be set.

Note: A transaction faulting at stage 2 is, when terminated, always aborted.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

It is IMPLEMENTATION DEFINED whether the SMMU supports the Stall model, the Terminate model, or both. Where system usage cannot be anticipated, ARM recommends that both fault models (SMMU_IDR0.STALL_MODEL == 0) and both termination models (SMMU_IDR0.TERM_MODEL == 0) are implemented.

If there is a risk that the stability of the system is compromised when the stall configuration is used for a set of client devices you can consider the following countermeasures:

- An implementation that supports both the Stall and Terminate models is permitted, but not required, to treat a stalling configuration for these devices as a terminating configuration.
    - For these devices, CD.S and STE.S2S are always taken to be zero, regardless of the value of the field.
- These devices are not required to be defined by the SMMU implementation, but are an IMPLEMENTATION DEFINED system property.

Note: For faulting transactions that are associated with client devices that have been configured to stall, but where the system has not explicitly advertised the client devices to be usable with the stall model, ARM recommends for software to expect that events might be recorded with Stall == 0.

Note: If CD.S and STE.S2S are treated as zero, the CD.R, CD.A and STE.S2R fields apply as specified in this section and section 5.5.

3.12.1 Terminate model

When stage 1 is configured to terminate faults, a transaction that faults at stage 1 is either terminated with an abort reported to the client device that is making the access, or the transaction completing successfully with reads returning 0 and writes being ignored (RAZ/WI), depending on the setting of CD.A and SMMU_IDR0.TERM_MODEL. See section 5.5.

When stage 2 is configured to terminate faults, a transaction that faults at stage 2 is terminated with an abort.

The behavior of the client device after termination is specific to the device.

If a stage that is configured to terminate faults is also configured with CD.R == 1 or STE.S2R == 1, as appropriate to the stage of the fault, the SMMU records the details of the access into one Event record in the Event queue, supplying information including:

- Address
- Syndrome
- Attributes (Read/Write, Inst/Data, Privileged/Unprivileged, NS)
- Type (S1/S2 Translation, Permission, Address Size, Access flag fault)

If the Event queue is full, the event record is lost.

Note: In some interconnects, stalling the transaction until its fault can be recorded might trigger interconnect timeouts or deadlocks from which it might be more difficult to recover than from a lost fault record. ARM expects that such fault records arise from programming errors and that software will not implement any mechanism that depends on the delivery of terminate fault records.

Streams that originate from PCIe subsystems must not stall and must be configured to use the Terminate model at all enabled stages of translation. This is enforced at stage 1 through the STE.S1STALLD flag, see section 16.4.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.12.2 Stall model

When a stage is configured to stall transactions on a fault, and a transaction experiences a Translation-related fault as described in 3.12, the faulting transaction does not progress and no response is reported to the client device. The transaction is buffered in a stalled state until subsequent resolution. The SMMU always records the details of the access into the Event queue. A stalled transaction is retried or terminated by issuing a CMD_RESUME or CMD_STALL_TERM command.

If retry is chosen, the transaction is handled as though it had just arrived at the SMMU. This means that the transaction will be affected by any configuration or translation changes that occurred since it originally stalled.

Note: This means a transaction can stall and later when it is retried, use a configuration that causes it to immediately terminate, for example, a change to stall configuration in the meantime. This property can safely clean up stalled transactions on a stream by ensuring that a new configuration for transactions that are retried causes them to be terminated.

If a stalled transaction is terminated by a CMD_RESUME command, a command parameter determines whether an abort is reported, or if SMMU_IDR0.TERM_MODEL == 0, whether the transactions completes with RAZ/WI behavior.

To ensure that no transaction is stalled indefinitely, software must ensure that every stall event has a corresponding CMD_RESUME command, is subject to a CMD_STALL_TERM command, or that stalled transactions are terminated because translation is disabled by clearing SMMU_(S_)CR0.SMMUEN to 0.

When an event record is generated for a stalled transaction, a Stall Tag (STAG) is supplied by the SMMU as part of the record to uniquely identify the transaction. The SMMU uses the combination of StreamID and STAG parameters to CMD_RESUME to identify the stalled transaction. A CMD_RESUME command has no effect on any stalled transaction other than on the transaction that is uniquely identified by the combination of STAG and StreamID.

Note: This identification is required for virtualization correctness, where a CMD_RESUME from a guest VM is trapped and reinterpreted by a hypervisor and generates a CMD_RESUME to the SMMU. The hypervisor validates the correctness of the StreamID parameter, but the STAG parameter is passed directly from the guest, and cannot be trusted to be correct and cannot be the sole selector of a stalled transaction.

The format of the STAG field is implementation specific, with the restriction that a value cannot be re-used until the transaction it was last associated with has been acknowledged through a CMD_RESUME or a CMD_STALL_TERM command, or translation is disabled by clearing SMMU_(S_)CR0.SMMUEN to 0.

If the Event queue is not writable at the time when the fault record of a stall is to be written, the stalled transaction is retried as though it had just arrived when the queue is next writeable and a new fault record is generated. For more information about recording faults and events, see section 7.2.

Software can depend on the delivery of fault records from stalled transactions, see section 3.12.4.

Note: Retrying the stalled transaction when the queue becomes writable might lead to the transaction succeeding or experiencing a different type of fault, if the configuration or translations were changed before the queue became writable. Therefore, an event can be written that is different to the originally-attempted event.

If the client device and interconnect rules allow it, a later transaction might pass through the SMMU and complete before an earlier stalled transaction that is associated with the same stream. The SMMU does not require any additional ordering between transactions from different streams beyond that required by the interconnect rules.

Note: The following cases are all considered to be from different streams:

- Transactions with different StreamIDs,
- Transactions with the same StreamID but different SubstreamIDs,
- Two transactions with the same StreamID but where only one transaction has a SubstreamID.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.12.2.1 Suppression of duplicate Stall event records

If a transaction faults and then stalls, and a subsequent transaction belonging to the same stream also faults and then stalls, the SMMU is permitted but not required to suppress the generation of a new stall fault record for the new transaction if all of the following apply:

- The transactions require access to the same page.
- The transactions have the same privilege.
- The transactions have the same data/instruction attribute.
- The transactions have the same type, that is they are both reads or both writes.
- The transactions are associated with the same SubstreamID, if present.
- The first stalled transaction is still stalled when a subsequent transaction stalls.

ARM recommends that an implementation suppresses additional fault records where possible.

Note: It is not guaranteed that event records are suppressed in all possible scenarios. Software must ensure correctness where a transaction records a fault that duplicates a previous fault that was recorded for an earlier transaction.

When a stall fault record is acknowledged by a CMD_RESUME command, any related suppressed stalled transaction are retried by the SMMU as though they had just arrived.

Note: A series of faults for one page might result in a single stall fault record, with a single CMD_RESUME command enabling all stalled transactions for that page to progress. If the CMD_RESUME command terminates the stalled transaction that is specified by the stall fault record, the re-trying of the other stalled transactions might cause new fault records to be recorded.

Note: For example, transactions A,B,C,D from the same stream that fault for the same reason might cause a single stall record for A to be recorded, and those for B,C,D to be suppressed. If software decides that the address was an error and terminates A, transactions B,C,D retry and fault again. A stall for B is recorded (and C,D might be suppressed). Software terminates B and the process repeats. Ultimately, A, B, C, D are all visible to software (rather than some being silently terminated), which can aid debug.

Stall fault records are not merged, see section 7.3.1.

Note: The suppression of identical stall fault records as described in this section is not the same as non-stall events being merged. When a stall record is suppressed, a stalled transaction still might exist and can affect future behavior, whereas the act of merging non-stall events completes the delivery of those events.

If a new transaction stalls for a reason that is unrelated to that of an existing stalled transaction, a new fault is recorded, – that is, it is not suppressed by dissimilar prior stalls even for the same StreamID and SubstreamID. ARM recommends that the new fault is recorded without being delayed by prior unrelated faults or CMD_RESUME activities where possible.

The SMMU does not record more than one fault for each incoming transaction, with the exception of the scenario in which a transaction stalls, and is explicitly retried with CMD_RESUME(Retry). After this command it is considered to be a new transaction and might again encounter a fault.

3.12.2.2 Early retry of Stalled transactions

The SMMU is permitted to speculatively retry a stalled transaction without first receiving a CMD_RESUME(Retry) command that matches the stalled transaction, this is referred to as *early retry*. If this occurs:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- An early retry is similar in function to the retry caused by an explicit CMD_RESUME(Retry). The transaction undergoes the full translation procedure and does not use any stale cached configuration or translation data that was invalidated since the time of the stall.
- A recorded stalled transaction causes a single fault record. An early retry of the stalled transaction does not cause additional faults to be recorded. When a retry is directly caused by a matching CMD_RESUME that indicates a retry, it is not considered to be an early retry, and this rule does not apply. This rule is in keeping with the behavior that an explicit retry command causes the transaction to be retried as though it had just arrived at the SMMU.
  Note: This rule includes the case where configuration has been changed to terminate faults after the transaction stalls then, under the new configuration, the transaction retries successfully by termination. Alternatively, if a retry occurs under stall fault configuration, the transaction remains stalled. Neither of these cases result in the reporting of a new event record.
- The progress of a transaction and the device-specific behavior are the only indications that an early retry has occurred that is visible to software.
- A successful early retry does not remove the requirement for software to acknowledge the stall fault record, see section 3.12.2. A successful early retry does not remove the restriction on re-using STAG values, see section 3.12.2. If targeted by CMD_RESUME(Terminate) or CMD_STALL_TERM, a stalled transaction is eventually terminated, if the transaction does not early-retry and successfully progress into the system before the termination can take place. If the transaction early-retries and fails to successfully translate, it remains stalled until the termination action takes effect, or a successive early-retry enables the transaction to progress successfully.

A CMD_RESUME(Retry) guarantees that the stalled transaction will be retried at a future point, unless it is terminated by CMD_STALL_TERM command or an SMMUEN transition before the retry. A stalled transaction is only guaranteed to be retried by the use of a CMD_RESUME(Retry) command.

A CMD_RESUME(Terminate) does not prevent a stalled transaction from being retried after the CMD_RESUME is consumed by the SMMU, but guarantees that the transaction will be terminated if the transaction cannot successfully early-retry.

Note: For example, if translations have not changed from the time that a fault was generated, a transaction cannot successfully early-retry.

Note: ARM does not expect software to modify a TTD from a faulting or invalid state into a valid state, and then terminate a transaction that has previously stalled because of the initial state of the TTD. The transaction could early-retry, observe the valid state and then progress into the system.

If the SMMU is able to successfully early-retry a stalled faulting transaction before the original stall event is committed to be written to the Event queue, the SMMU is permitted to discard the fault event or to continue on and commit to the event write.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: To software, this race condition is indistinguishable from a temporally-later transaction that translates successfully the first time so a stall event record is not required. If an implementation records the event, the behavior described in this section applies.

If a stalled faulting transaction is retried before the original stall event is committed to be written to the event queue and experiences a fault that is different to the previous fault, the most recent fault is recorded, provided that it is possible to do so and that the previous fault is invisible.

Note: This scenario is permitted to occur when the Event queue is writable.

See section 7.2.1 for retry behavior requirements when the Event queue is not writable.

3.12.2.3 Miscellaneous Stall considerations

The number of transactions that can be stalled before the ingress port cannot accept any more transactions, from the same stream or from other streams, is implementation specific. Stalling traffic can therefore cause backpressure that affects the flow of traffic for other devices behind the SMMU.

If a stall blocks other traffic and resolving the fault condition that caused the stall involves transferring data using another device, the system architecture must ensure that the act of fetching the data will not itself stall behind the original transaction.

Note: STE.S1STALLD == 1 prevents a guest VM from using the Stall model. This guarantees that stalled transactions cannot affect other parts of the system, such as a different guest VM, where stalls could cause deadlocks. ARM expects that hypervisor software uses the virtualization of SMMU_IDR0.STALL_MODEL to report to the guest VM that the Stall model was not supported.

If a transaction experiences a fault during an IPA to PA translation of a stage 1 translation table walk or CD fetch, it is not required to be terminated and might stall, depending on the stage 2 fault configuration provided by STE.S2S. Software might address the cause of the stage 2 fault and retry the transaction, which will re-fetch the configuration and translation structures as necessary.

3.12.3 Considerations for client devices using the Stall fault model

If a transaction from a client device experiences a fault that stalls and is terminated by software issuing CMD_RESUME(Terminate), the transaction is marked and guaranteed to terminate at some point in the future if translations do not change so as to allow an early-retry to succeed in the meantime. The SMMU does not guarantee when a stalled transaction is terminated.

Note: A situation might arise in which software is required to reconfigure translations so that a previously-marked stalled transaction might now succeed if it were to retry. For example, a transaction that is made to an unmapped address causes an initial fault, and then a terminate operation is performed. Later software creates a legitimate mapping at that address and, if the original transaction was a write that retries and now succeeds, data corruption might result. Software might need a mechanism to ensure that previous transactions have all completed, both terminated stalls and transactions that are progress that the SMMU is not yet aware of.

The system, or client devices, must provide a mechanism to enable software to wait for these previous transactions to complete before changing configuration to a state that might let them proceed. This might be an explicit indication from the client device that its outstanding transactions have all been terminated or completed, an interconnect ordering guarantee that prior transactions are all visible, or another mechanism.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.12.4 Virtual Memory paging with SMMU

The SMMU architecture supports three models of usage with respect to translation-related faults that occur during translation of client device accesses:

1. A fault that occurs due to a device access might always be considered to be an error by the system and is terminated.

   Note: This might be the result of a programming error.

2. A fault that occurs due to a device access might be considered permanent due to a programming error, or temporary due to particular page state resulting from use of virtual memory with the address space, and one of the following is configured to occur:

   a. The device transaction is stalled, the fault is reported to software and then the transaction is resumed after the virtual memory system resolves the cause of the fault. Or, if the virtual memory system determines that the access was invalid, the transaction is terminated. This model can only be used with a device and interconnect that can support stalls.

   b. For PCIe devices, for which transactions cannot safely be stalled, the PCIe specification provides ATS and PRI. ATS enables an endpoint to ascertain whether a page can be accessed without causing a fault in the SMMU before accessing it. PRI provides a mechanism for a page fault to be resolved if the prior ATS step indicates a fault would otherwise occur.

3.12.4.1 Page-in request event

When non-PCIe devices are used with the Stall fault model to access paged virtual memory spaces, the Stall fault record itself is the notification to software that a page miss occurred and that software intervention is required.

Note: Devices used with, or integrating, an SMMU will generally emit transactions when the access is required. Although read speculation is permitted, writes cannot be emitted speculatively to trigger a page fault early, see section 3.14. In particular, stall fault records do not arise from accesses marked speculative.

An optional hint event record, E_PAGE_REQUEST, can be provided by an implementation to request that software initiates any costly page-in operations early. An implementation might provide an IMPLEMENTATION DEFINED mechanism to convey this message from client devices. This message:

- Is a hint, and can be ignored or dropped by the SMMU or software.
- Can be issued speculatively.
- Requires no response.

Note: A stall fault record is generated in response to a non-speculative transaction. A speculative transaction generates no software-visible record. E_PAGE_REQUEST allows a software-visible record to make an early start on fetch of pages from secondary storage and can be used to hide latency.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.12.5 Combinations of fault configuration with two stages

When the Stall model and the Terminate model are used differently at different stages of translation, the resulting behavior of depends on the stage at which the transaction faulted and the type of fault. For Translation-related faults that can stall the following scenarios arise:

Stage 1 config	Stage 2 config	Fault at	Transaction result	Event parameters	Hypervisor behavior
Terminate	Terminate	Stage 1	Terminated	VA	Event passed to guest as a stage 1-only event
		Stage 2	Terminated	VA, IPA	Might log IPA of fault for debug purposes. (1) Might pass event to guest.
Terminate	Stall	Stage 1	Terminated	VA	Event passed to guest as S1-only event
		Stage 2	Stalled	VA, IPA	May terminate with CMD_RESUME(Terminate) and log IPA of fault for debug purposes. Or, correct translation for IPA then CMD_RESUME(Retry). (1) Might pass event to guest if terminated.
Stall	Terminate	Stage 1	Stalled	VA	Event passed to guest as stage 1-only event with stall. Guest must RESUME(Retry/Terminate)
		Stage 2	Terminated	VA, IPA	Might log IPA of fault for debug purposes. (1) Might pass event to guest.
Stall	Stall	Stage 1	Stalled	VA	Event passed to guest as S1-only event with stall. Guest must CMD_RESUME(Retry/Terminate)
		Stage 2	Stalled	VA, IPA	Might terminate with CMD_RESUME(Terminate) and log IPA of fault for debug purposes. Or, correct translation for IPA then CMD_RESUME(Retry). (1) Might pass event to guest if terminated.

1. Might pass event to guest: Anything that is terminated at stage 2 is equivalent to a stage 1 external abort. A successful stage 1 translation that outputs an incorrect IPA that leads to a stage 2 fault would not ordinarily be reported to the guest through its SMMU interface, because its stage 1 translation succeeded

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

and the error arises outside of the (stage 1) domain of the SMMU interface. ARM expects that a stage 1 translation table walk that faults at stage 2 is reported to the guest as F_WALK_EABT by the hypervisor.

All other fault types cause the transaction to be aborted. For example, a failure to locate a valid STE (F_BAD_STE) or CD (F_BAD_CD) terminate the transaction with an abort.

Note: When both stage 1 and stage 2 are enabled, a CD or stage 1 translation table descriptor fetch might cause a stage 2 Translation-related fault, and might therefore stall the transaction. Regardless of the reason for making the IPA access, the fault can be resolved at stage 2 and restarted. This is the same behavior as with a faulting IPA access for the transaction address after stage 1 translation.

3.13 Translation table entries and Access/Dirty flags

The SMMU might support hardware translation table update (HTTU) of the Access flag and the Dirty state of the page for AArch64 translation tables.

Some ARMv8-A PEs might support *Hardware Update to Access flag and Dirty state* [5]. SMMU support of HTTU can coexist with both hardware and software flag update from the PEs. The SMMU update of descriptors behaves in an identical manner to those described in [5], with the additional SMMU-specific behavior in section 3.13.4, although its configuration method differs.

HTTU increases the efficiency of maintaining Access flag and Dirty state in translation tables. A single translation table can be shared between any combination of agents that perform software updates of flags, and other agents that perform HTTU. Agents supporting HTTU update the flags atomically. Software must also use atomic primitives to perform its own updates on translation tables when they are shared with another agent that performs HTTU.

Note: In general, an update of the Access flag and the Dirty state of the page in a system is associated with the use of dynamic paging and, in the context of the SMMU, associated with DMA targeting paged memory. ARM does not expect applications that constrain DMA access to static, pinned or non-paged mappings to perform or require dynamic update to the Access or to the Dirty state of the page.

Support for HTTU is indicated by SMMU_IDR0.HTTU, and can be one of the following:

- No flag updates are supported.
- Only Access flag updates are supported.
- Update of both the Access flag and the Dirty state of the page is supported.

If HTTU is supported, separate enable bits in the CDs and STEs determine whether a particular stage 1 or stage 2 translation table (referenced from the CD or STE) is updated in this manner, and the scope of the updates.

Note: It is possible for several CDs to reference the same translation table, or for several STEs to reference the same CD. Where translation tables are shared between CDs that contain the same ASID (within a translation regime), the CD HA and HD fields must be identical. See section 5.4.1.

Note: *Accessed* means a translation to which an access has been made. Software might attempt to detect a working set by clearing the Access flags and observing which flags are set again. *Dirty state of the page* means a writable translation to which a write access or other modification has been made. When reclaiming or repurposing a Dirty page, software might preserve the modifications to storage. *Clean* means a writable translation to which a

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

modification has not been made. When reclaiming or repurposing a Clean page, software might simply discard the page contents (as another up-to-date copy might be available in storage elsewhere).

3.13.1 Software update of flags

Note: In the context of a PE that does not support HTTU, software is generally expected to maintain the Access flag and the Dirty state of a page, where required, as follows:

- A read or write that fetches a TTD with AF == 0 causes an Access fault, if AFFD == 0. An exception handler marks the TTD as AF == 1 and retries the instruction that caused the access. Agents are not permitted to cache such entries in TLBs. No TLB invalidation is required when setting AF to 1.
- A Dirty flag is usually implemented in software by write-protecting a translation. A write access to such a translation generates a Permission fault, at which point the exception handler might rewrite the TTD to mark it writable. The exception handler might use additional software structures or a software-defined TTD flag to differentiate a genuinely non-writable page from a page that is only temporarily non-writable in order to generate the Permission fault. In this arrangement, a TTD that has write permission is considered to be *Writable, Dirty* and a TTD that has no write permission but is marked as temporarily unwritable is considered to be *Writable, Clean*.
- A write to a genuinely non-writable page is an error. A write to a Writable Clean (temporarily non-writable) page causes the page to become Writable Dirty. A write to a Writable Dirty page causes no additional state change.
- An Access fault takes priority over a Permission fault. That is, a write to a Writable Clean page with AF == 0 and AFFD == 0 causes an Access fault, and only after AF == 1 does a Permission fault occur.

When pinned DMA translations are used with an SMMU, software can update the translation flags as appropriate to the expected access. ARM does not expect faults to be generated when pinned translations are used, and such faults represent a programming error. ARM expects software to use the Terminate model for such scenarios, so that faulting transactions are terminated.

An SMMU can operate in a similar manner to the PE example when using unpinned DMA translations, so that transactions that are translated by the SMMU cause faults to be recorded and the SMMU driver software sets TTD state in response to these records. For more information about faults, see section 3.12.

Where this is the case, ARM recommends that this is implemented using the Stall model. ARM expects that the SMMU driver maintains software Access or Dirty state by doing one of the following:

- Responding to an F_ACCESS fault by setting AF to 1 in the relevant TTD.
- Responding to an F_PERMISSION fault for write to a Writable Clean page by marking the page as Writable Dirty.
- Finally, issuing a CMD_RESUME to the SMMU to retry the transactions held up due to the fault

Note: The AFFD field in a stage 1 and stage 2 configuration modifies the behavior of the AF flag of a TTD. A TTD, at a translation stage with AFFD == 1, and AF == 0 does not cause an F_ACCESS to be generated. Instead, the translation is used as though AF == 1. This configuration is only relevant where HTTU is not used.

A translation table can be shared between multiple agents, if all agents that update the Access flag and the Dirty state of the page use the same semantics to differentiate a TTD marked non-writable from one marked temporarily non-writable. Usually, this is a software-defined bit that flags a page as potentially writable as opposed to a page that is intended to always be non-writable.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

HTTU removes the fault record and software handling from the path of updating translation table flags. An agent is permitted to perform HTTU on a translation table that might be shared with an agent performing software update. The Dirty Bit Modifier field has been added in ARMv8.1-A to differentiate non-writable and Writable Clean states. For more information about the Dirty Bit Modifier, see section 3.13.3. Software intending to provide software-updated TTD flags from one agent (for example a PE without HTTU) while sharing translations with another agent that uses HTTU must use the DBM flag convention, and perform atomic updates.

3.13.2 Access flag hardware update

When HTTU is supported and enabled for a stream, a translation that causes an SMMU fetch of a descriptor with AF == 0 that would, without HTTU and with AFFD == 0, have caused an Access fault performs an atomic update to set AF == 1 in the descriptor. The SMMU never clears AF.

If access to a descriptor causes a permission fault, it is UNKNOWN whether the AF flag of the descriptor is updated to 1.

When HTTU is disabled, or not supported by the SMMU, a transaction that leads to access of translations with AF == 0 and AFFD == 0 causes F_ACCESS.

If the update of the Dirty state of the page takes place, the final TTD will also have AF == 1.

3.13.3 Dirty flag hardware update

In order to coexist with an agent that is not using hardware update, HTTU defines a new flag, at bit[51] of block and page TTDs, called the *Dirty Bit Modifier* (DBM). The DBM bit marks the overall intention of a translation as ultimately writable, to differentiate a non-writable page from a Writable Clean page in the same way as a software-maintained mechanism would.

Note: The ARMv8-A stage 1 TTD field AP[2:1] has no bit[0]. The stage 2 equivalent is named S2AP[1:0]. Note that earlier PE architecture referred to HAP[2:1], which is the same field as S2AP[1:0].

When HTTU of the Dirty state of a page is supported and enabled for the stream, a non-writable TTD is automatically marked as writable by the SMMU when a translation for a write occurs, if it is a TTD with DBM == 1. A Permission Fault is not generated, and the translation continues.

Specifically, if a TTD is read-only only as a result of AP[2:1] == `0b1x` at stage 1, or non-writable using S2AP[1:0] == `0b0x` at stage 2, then if DBM == 1 and a translation for write occurs, the SMMU atomically sets AP[2] to 0/HAP[2] to 1 in the TTD held in memory, in a coherent manner if appropriate. If DBM == 0, the page has no write permission and a write translation results in a Permission fault.

Note: HTTU of the Dirty state of a page is not applicable to a TTD that is made effectively read-only because of the hierarchical control of access permissions using APTable. All references to page or block permissions in this section are made on the assumption that the page or block is otherwise accessible, will not generate a Permission fault for other reasons, such as PAN or the APTable having removed access, and that a page is only read-only (or otherwise non-writable) because of the page or block AP/S2AP permissions.

When the APTable does not remove write access:

- A read-only stage 1 TTD has DBM == 0 and AP[2:1] == `0b1x`. A non-writable stage 2 TTD has DBM == 0 and S2AP[1:0] == `0b0x`.

  A write causes a Permission fault as the page has no write permission.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The software fault handler invokes an error-handling routine. Because DBM == 0, the software handler can determine that the page is not allowed to be written. In the case of an SMMU stalled fault, software can use CMD_RESUME to terminate an erroneous transaction.

- A Writable Clean TTD has DBM == 1 and AP[2:1] == `0b1x`/S2AP[1:0] == `0b0x`.

  Without HTTU, this TTD is non-writable and a write causes a Permission fault. Because DBM == 1, the page is intended to be writable and the software fault handler can mark the page as dirty by setting AP[2] == 0/S2AP[1] == 1 and performing the appropriate TLB invalidation. The page is now marked Writable Dirty. In the case of an SMMU stalled fault, software can use CMD_RESUME to retry the transaction, which might then continue without fault.

  When HTTU is enabled, a write transaction causes the SMMU to atomically set AP[2] == 0 or S2AP[1] == 1 as appropriate, and then allows the write to proceed.

- A Writable Dirty TTD has AP[2] == 0/S2AP[1:0] == `0b1x`.

  With or without HTTU, this page is writable and will not generate a Permission fault on a write.

  Note: Although DBM is ignored by hardware in this state, it might be useful for software to use the convention of leaving DBM == 1 when a page AP[2] or HAP[2] transitions from non-writable to writable. This allows the DBM bit to be used as a one-bit flag to indicate that, overall, a page is intended to be written regardless of its current Clean or Dirty state.

- The SMMU never sets or clears DBM.
- The SMMU never clears S2AP[1].
- The SMMU never sets AP[2]. A TTD is never made writable by the SMMU unless DBM == 1.

3.13.4 HTTU behavior summary

SMMU HTTU operation has the same behavior as described in *Hardware Updates to Access Flag and Dirty State* in the ARMv8.1-A architecture [5].

The following HTTU behavior is specific to the SMMU:

- A TTD update that occurred because of a completed ATOS translation is made visible to the required Shareability domain, as specified by the translation table walk attributes, by completion of a CMD_SYNC that was submitted after the ATOS translation began.
- A TTD update that occurred because of a completed incoming transaction is made visible to the required Shareability domain (as specified by the translation table walk attributes) by completion of a CMD_SYNC that was submitted after the completion of the incoming transaction.
  - In addition, the completion of a TLB invalidation operation makes TTD updates that were caused by transactions that are themselves completed by the completion of the TLB invalidation visible. Both broadcast and explicit CMD_TLBI_* invalidations have this property.

    Note: The SMMU HTTU behavior follows the same rules as ARMv8.1-A [5], including all TLB invalidation completion requirements on HTTU visibility from VMSA.

3.13.5 HTTU with two stages of translation

When two stages of translation exist, multiple TTDs determine the translation, that is the stage 1 TTD, the stage 2 TTDs mapping all steps of the stage 1 walk, and finally the stage 2 TTD mapping the IPA output of stage 1. Therefore one access might result in several TTD updates. Figure 12 shows an example:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

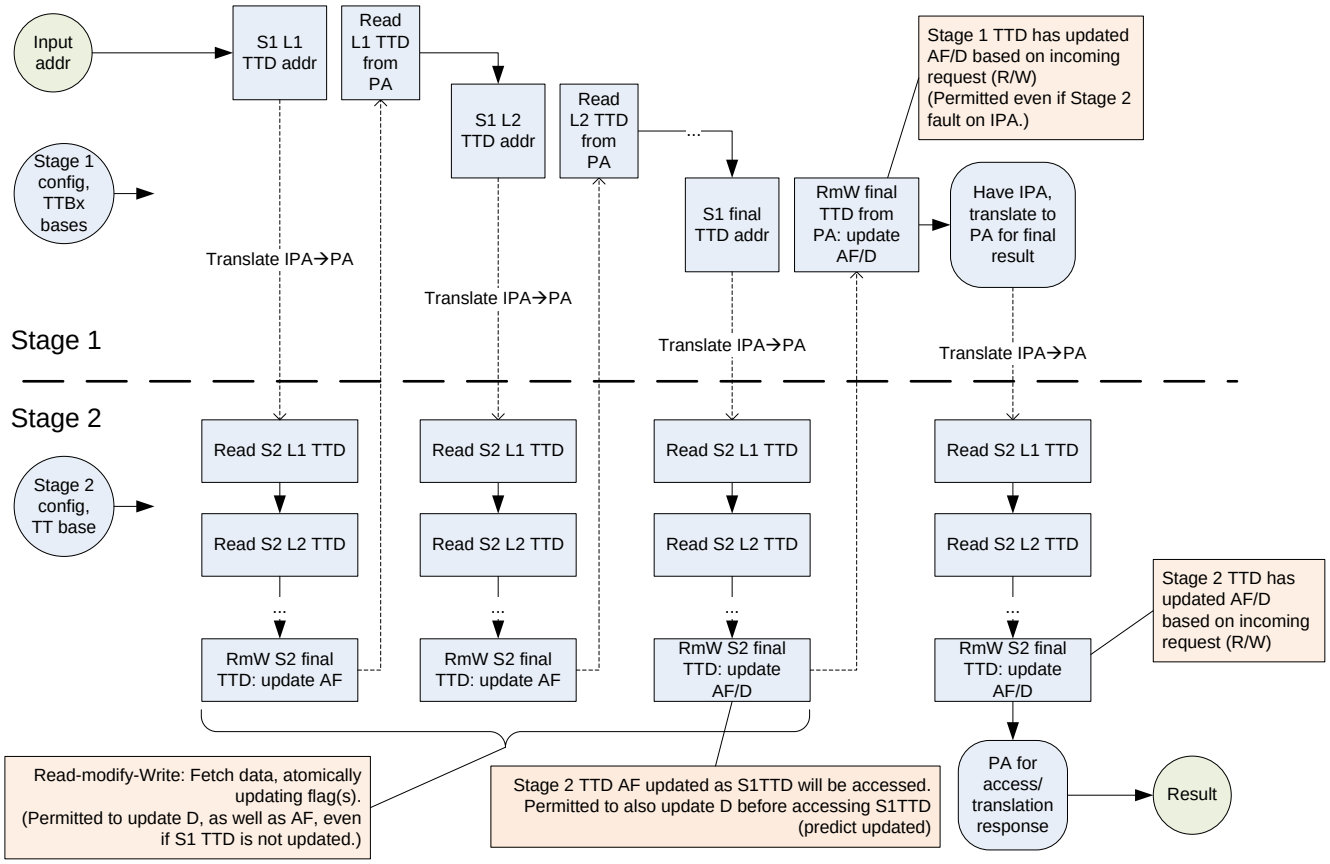


Figure 12: Example Hardware flag update with nested translation

Note: Figure 12 is an example procedure and does not depict all permitted ways of performing a nested translation walk with HTTU enabled.

Because a stage 1 TTD hardware update is a write, the stage 2 mapping for its IPA must allow writes for the update to succeed.

3.13.6 ATS, PRI and translation table flag update

When ATS and PRI are used to support device access to dynamically paged memory, the Access state and the Dirty state of the page need to be maintained. This section describes the SMMU page flag maintenance behavior in a system using ATS with PRI targeting dynamically paged memory.

Note: Maintenance of the Access flag and the Dirty state of the page is primarily of importance to DMA to unpinned or paged memory, because use-cases with DMA to pinned memory would normally statically initialize page state.

3.13.6.1 Hardware flag update for ATS & PRI

Because the purpose of ATS is to cache translations outside the SMMU and to avoid subsequent translation interaction with the SMMU, if HTTU is enabled it is performed at the time of the ATS Translation Request (TR).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When an ATS TR is made, it must be assumed that a device will subsequently access the page. If the page is otherwise valid and an ATS response will be returned, AF is set to 1 in the descriptor in the same way as a direct transaction access through the SMMU.

In addition to the behavior that is described earlier in this section, if hardware-management of Dirty state is enabled and an ATS request for write access (with NW == 0) is made to a page that is marked Writable Clean, the SMMU assumes a write will be made to that page and marks the page as Writable Dirty before returning the ATS response that grants write access. When this happens, the modification to the page data by a device is not visible before the page state is visible as Writable Dirty. If HTTU is only enabled for Access, an ATS request for a write to a Writable Clean or Read Only page results in an ATS Translation Completion with W == 0, and write access is denied.

If an ATS Translation Request is made for a write (NW == 0) to a nested translation configuration and the associated stage 1 translation is read-only, the Dirty state is not updated in either of the stage 1 or stage 2 TTDs.

Note: This also applies if the stage 2 TTD is writable.

When HTTU is enabled for stage 1 and stage 2 and Split-stage ATS is used (STE.EATS == `0b10`), the ATS TR performs HTTU at stage 1, and updates for the stage 2 TTDs that are used to fetch and update the stage 1 TTDs are made. The following applies to the stage 2 TTD for the final IPA:

- The AF in the stage 2 TTD for the final IPA is permitted to be speculatively set to 1 by the ATS TR.
- If write permission is granted in the ATS Translation Completion, a Writable Clean stage 2 TTD for the final IPA is permitted to be marked as Writable Dirty by the ATS TR.
    - When a subsequent Translated access to the IPA is translated, this choice does not affect the HTTU behavior of stage 2.
    - This choice does not affect the permissions that are returned in the Translation Completion, which reflect whether a write transaction is permitted by the combined permissions of both stages and treat a Writable Clean stage 2 TTD as writable. See section 13.6.3.
- The stage 2 translation of a subsequent Translated access marks the stage 2 descriptor as Accessed and might mark the descriptor as Writable Dirty, provided that the ATS TR has not already performed this action.

3.13.6.2 Behavior with respect to flag maintenance for ATS & PRI without HTTU

If HTTU is not enabled for the Access flag, an ATS request to a page with AF == 0 and AFFD == 0 is denied. For this address a response granting R == W == 0, that is no access, is returned. The client device might then raise an error in a device-specific manner, or might issue a PRI page request, if supported and configured, to request that software makes the page available. Software can manually set AF == 1 on receipt of the PRI page request in anticipation of the device access.

An ATS request to any read-only page does not grant write access, that is it returns W == 0, if hardware update of Dirty state of the page is not enabled. Read access might be granted in the response, if the conditions for the Access flag set out in this section are satisfied. The client device might raise an error in a device-specific manner, or might issue a PRI page request to request write access to the address. On receipt of a PRI request, software could assume that a request issued for write was initiated because data will shortly be written and mark the page Writable Dirty before responding to the PRI request.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

An ATS request for write to a page marked writable might grant write access, that is it returns W == 1 in the response. Software must consider writable pages as potentially dirty.

Note: PCIe PRI requests can be issued speculatively by an Endpoint. This implies speculatively marking the page as Dirty. This is not permitted by the ARMv8-A architecture [6] and might be problematic for some software systems.

Because pages cannot speculatively be marked as Dirty, ARM recommends that a system designed for general-purpose software supports HTTU when PRI is used, so that the state of the page is marked Dirty only when a request for write access is made using ATS.

3.13.7 Hardware flag update for Cache Maintenance Operations and Destructive Reads

HTTU for Dirty bit update is not performed for the following operations:

- Invalidate Cache Maintenance Operations. See section 16.7.2.2 for more information.
- Destructive Reads. See section 3.22.2 for more information.

When these operations are performed to a Writable Clean TTD, the TTD is not updated to be Writable Dirty. If the required Read or Execute permissions are available, but the TTD is not Writable Dirty, the operations are downgraded as described in the corresponding section.

This rule does not affect HTTU of the Access flag, which occurs if required.

Note: For the purposes of determining execute permission, a Writable Clean TTD is considered to be writable when HTTU is enabled, which is consistent with the ARMv8-A architecture. As described in this section, this principle applies even when the TTD is not updated to Writable Dirty for an Invalidate or DR.

3.14 Speculative accesses

An implementation might allow incoming transactions to be marked as speculative in an IMPLEMENTATION DEFINED manner. Only read transactions are allowed to be marked as speculative. The behavior of a write transaction that is marked as speculative is always to terminate the transaction with an abort, and no event is recorded to software.

The behavior of a read transaction that is marked as speculative depends on two things:

1. If the translation occurs successfully without faulting, the read transaction continues into the system and returns data. Otherwise if any kind of fault or configuration error occurs, the transaction is terminated with an abort; no event is recorded to software for any speculative transaction. The determination of a fault is no different to non-speculative read transactions, including Access flag faults.
2. If HTTU is enabled and translation succeeds without fault, the read transaction updates the Access flags of relevant TTDs.

The SMMU HTTU rules match those set out for ARMv8.1-A [5], with respect to hardware update of translation table descriptor access/dirty flags, including update of stage 2 translation table flags for both speculative accesses made at stage 1 and writes of stage 1 descriptors due to the setting of Access flags.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

An implementation might provide translation services to a client device, and might support speculatively-issued Translation Requests. An IMPLEMENTATION DEFINED mechanism must be used to differentiate speculative Translation Requests from non-speculative Translation Requests.

Note: This mechanism might arise as an implementation-specific service provided to another device. PCIe ATS Translation Requests are always non-speculative.

If a received Translation Request is marked as speculative, behavior is dependent on the read/write property of the request:

- Translation Requests for an address to be written grant write in the response only if the TTDs that translate the address are all marked Writable Dirty. If hardware management of the Access flag is enabled, such a request updates AF. If hardware management of Dirty state is enabled, speculative Write Translation Requests do not mark any Writable Clean TTD in the first or only stage of translation as Writable Dirty. If the TTD is marked Writable Clean, the response does not grant write access.
- Translation Requests for an address to be read return a successful response, if appropriate, and if hardware management of the Access flag is enabled updates AF.
- In both cases, if hardware management of Access flag and Dirty state is enabled in a nested translation then an update of a stage 1 TTD to set AF or the Dirty state of the page might cause the stage 2 TTDs related to the updated stage 1 TTD to be marked as Dirty as required.

The response to a Translation Request indicates whether a translation request was denied because of a page fault or otherwise missing translation, or whether a valid translation existed but the request failed because the translation was Writable Clean.

Note: A device might use this information to determine whether to stop making requests or whether to subsequently try again with a non-speculative write.

For speculative accesses of SMMU structures and translations, see section 3.21.1 and 3.21.3.

3.15 Coherency considerations and memory access types

ARM anticipates that the SMMU will access all in-memory structures and queues in a manner that does not require software cache maintenance of the PE caches. ARM expects that this be IO-coherent access to normal shared memory, but in implementations that cannot support cache coherency, this might be non-cached access. Some embedded implementations might require use of memory mapped as non-cached by the PEs, see section 3.16 below. The degree to which the different memory access types and attributes are supported is IMPLEMENTATION DEFINED.

All in-memory structures and queues are accessed using Normal memory types. Configuration fields exist for Stream table, Context descriptor and translation table fetches to govern Cacheability and Shareability of such accesses. MSI writes can be configured to make Device-type accesses.

If hardware update of the Access flag or the Dirty state of a page is supported, atomic access is required to update translation tables that are shared between the PE and SMMU. Support for atomic access using local monitors requires a fully-coherent interconnect port.

If the memory system supports ARMv8.1 [5] atomic operations, the SMMU might support atomic updates without local monitors, and not require a fully-coherent port. Because different SMMU implementations might use different mechanisms for atomic update of the flags, and because local monitors require coherent cacheable access, behavior is IMPLEMENTATION DEFINED if hardware flag updates are enabled on translation tables configured to be accessed as Non-cacheable.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

To limit complexity, the SMMU might respond to snoops from the system only as much as required for atomic updates to translation tables with local monitors, if required. This means that all other memory access by the SMMU might be IO-coherent. That is, SMMU configuration caches are not required to be snooped by PE accesses. When configuration data structures are changed, software is required to issue invalidation commands to the SMMU.

The SMMU respects the same single-copy atomicity rules as PEs regarding 64-bit translation table descriptor accesses.

When configured by software, that is when not fixed in embedded implementations, ARM recommends that the in-memory data structures and queues are treated as Normal memory cached by the PE when the SMMU implementation is able to access them IO-coherently.

Note: This might be useful to avoid explicit cache maintenance on the PE side. When an SMMU is not able to make IO-coherent access, a similar programming model might be achieved using normal non-cached mappings from the PE.

Note: The configuration structure invalidation commands might be used by a hypervisor to maintain coherency between guest and shadow structures that it might use.

When a system supports IO-coherent accesses from the SMMU for access to configuration structures, translation tables, queues and CMD_SYNC, GERROR, Event queue and PRI queue MSIs, this is presented to software using SMMU_IDR0.COHACC == 1. If a system does not support IO-coherent access from the SMMU, SMMU_IDR0.COHACC must be 0.

3.15.1 Client devices

SMMU translation of coherency traffic for client devices is not supported. Cache and TLB-maintenance operations sent from client devices into the system are not supported. These operations terminate in the SMMU.

Devices connected behind an SMMU cannot contain caches that are fully-coherent with the rest of the system because snoops relate to physical cache lines. Devices might contain caches that do not support hardware coherency and which might be filled using non-physical addresses through an SMMU.

However, a client device that contains a TLB filled from the SMMU might maintain a fully-coherent physically-addressed cache, using the TLBs to translate internal addresses to physical addresses before performing cache accesses. Such a case might arise where an SMMU is implemented as part of a complex device.

In distributed systems, different client devices might have different paths through the SMMU into the system, and these can differ in their ability to perform IO-coherent access. These paths might also differ from those used by the SMMU for its own configuration access, hence SMMU_IDR0.COHACC does not indicate whether client devices can also make IO-coherent accesses. ARM recommends that whether a given client device is capable of performing IO-coherent access is described to system software in a system-specific manner.

3.16 Embedded implementations

Some implementations might support the use of on-chip or internal storage for one or more of the Stream table structures, the Command queue, or the Event queue. This manifests itself as register base pointers and properties that are hard-wired to point to the on-chip storage. Such queues and structures are of a fixed size and

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

configuration and in all cases are discoverable by system software. Software must not assume that it is necessary to allocate tables in RAM and set up pointers. It must initially probe for an existing configuration. SMMU_IDR1.TABLES_PRESET and SMMU_IDR1.QUEUES_PRESET indicate that the Stream table base address and queue base addresses are hardwired to indicate pre-existing storage for the tables or queues, or both. When SMMU_IDR1.REL is set, the base addresses are given relative to the start of the SMMU register memory map, rather than as absolute addresses.

An implementation using internal storage for configuration and queues is not required to access this storage through the coherency domain of the PEs. Data accesses from the PE require manual cache maintenance or use of a non-cached memory type for these addresses.

3.16.1 Changes to structure and queue storage behavior when fixed/preset

Non-preset tables/queues are stored in normal memory. When an embedded implementation contains a preset structure or queue in internal storage it is not required that all bits of all structures/queue entries are accessible exactly as they would be in normal memory. For example, an implementation might not provide storage for fields in structures and queues that would not be used by architected behavior.

3.16.1.1 Event Queue and PRI Queue

All entries in an embedded Event queue or PRI queue, that is where SMMU_IDR1.QUEUES_PRESET == 1, are permitted to have read-only/write-ignored behavior with respect to software accesses.

3.16.1.2 Command Queue

Entries in an embedded Command queue, that is where SMMU_IDR1.QUEUES_PRESET == 1, are readable and writable, but are not required to provide storage outside of the union of all defined fields for all implemented commands. In addition, referring to the Command encodings in section 4, storage is not required to be provided for:

- Reserved and undefined fields.
- High-order bits of StreamID fields beyond the implemented range of StreamIDs.
- High-order bits of SubstreamID fields beyond the implemented range of SubstreamIDs (including the entire field if SubstreamIDs are not implemented).
- SSV fields, if SubstreamIDs are not implemented.
- STAG bits that are always generated as ‘0’ to software.
  Note: An implementation might choose to use fewer than 16 bits of STAG when communicating stalled faults to software.
- SSec, if Security is not supported.
- CMD_SYNC MSIData, MSIAddress and MSIWriteAttributes if MSIs are not supported by the Security state of the Command queue.
- ASID[15:8] if SMMU_IDR0.ASID16 == 0, or VMID[15:8] if SMMU_IDR0.VMID16 == 0.
- CMD_CFGI_STE Leaf parameter (embedded Stream tables are single-level).
- Fields in any command type that gives rise to CERROR_ILL.

A bit that is not stored due to these rules has RAZ/WI behavior.

Note: An implementation determines the set of required storage bits from implementation-specific configuration options and values.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Software must not assume that it can write an arbitrary 16-byte sequence to a Command queue entry and read back the sequence unmodified. However, functional fields that form valid command parameters must be readable by software for debug and read-modify-write construction of commands (the queue is not considered write-only).

3.16.1.3 Stream Table Entry

Entries in an embedded Stream table are freely read/write accessible, but storage is not required to be provided for:

- Undefined fields.
- Reserved/ RES0 fields.
- Fields that are IGNORED in all possible configurations that an implementation supports.
- Fields permitted to have RAZ/WI behavior.

As an example, storage is not required for STE.S1ContextPtr on an SMMU that has an embedded Stream table but does not support stage 1.

Note: Fields reserved for software use do not alter SMMU function but must be stored in their entirety.

3.17 TLB tagging, VMIDs, ASIDs and participation in broadcast TLB maintenance

Cached translations within the SMMU are tagged with:

- A translation regime, given by the STE's StreamWorld and derived in part from STE.STRW. This applies to all cached translations.
- An ASID, if the translation regime supports ASIDs.
- A VMID, if stage 2 is implemented by the SMMU and if the translation regime supports VMIDs)

This is summarized in the table below:

StreamWorld	Address type	Tags		
		ASID	ASET	VMID
NS-EL1	VA	Yes, if TTD.nG == 1	Yes [2]	Yes [1]
	IPA [1]	No	No [3]	Yes [1]
EL2	VA	No	No [3]	No
EL2-E2H	VA	Yes, if TTD.nG == 1	Yes [2]	No
Secure	VA	Yes, if TTD.nG == 1	Yes [2]	No
EL3	VA	No	No [3]	No

(1) If SMMU_IDR0.S2P == 1.

(2) ASET is required to be included in TLB records when TTG.nG == 0. ARM expects, but does not require, ASET to be included in TLB records when TTD.nG == 1 to support the limitation of broadcast invalidation feature of ASET.

(3) ARM permits but does not require the inclusion of ASET in TLB records for EL3 and EL2.

This is consistent with TLB tagging in PEs.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: In this document, the term *cached translations* refers to the contents of a PE-style ASID/VMID/Address TLB. Any cached configuration structures are considered architecturally separate from the translations that are located from the configuration. Configuration caches are not required to be tagged as described in this section.

Use of these tags ensures that no aliasing occurs between different translations for the same address within different ASIDs, or between the same ASID under different VMIDs, or between the same ASID within different translation regimes, or between different translation regimes without ASIDs (for example EL2 and EL3). For both lookup and invalidation purposes, ASID values can be considered to be separate namespaces within each VMID and translation regime. For example, TLB entries tagged as ASID 3 in a Secure stage 1 cannot be matched by lookups for ASID 3 in an NS-EL1 stage 1 configuration. Similarly, a TLB entry that is tagged as EL2 can never be matched by a lookup from an EL3 context, even if the address matches.

When SMMU_IDR0.S1P == 1, the SMMU supports 16-bit ASIDs if SMMU_IDR0.ASID16 == 1.

When SMMU_IDR0.S2P == 1, the SMMU supports 16-bit VMIDs if SMMU_IDR0.VMID16 == 1.

Consistent with PEs, all TLB entries inserted using NS-EL1 configurations are tagged with VMIDs when stage 2 is implemented, regardless of whether configuration is stage 1-only, stage 2-only or stage 1+stage 2 translation. When stage 2 is configured, with or without stage 1, or if stage 1-only translation is configured, the VMID is taken from the STE.S2VMID field. When stage 2 is not implemented, no VMID tag is required on TLB entries.

Note: Some implementations and interconnects might support the transmission of VMID value onwards into the system, so that slave devices might further arbitrate access on a per-transaction basis. Where SMMU bypass is enabled (SMMU_(S_)CR0.SMMUEN == 0) so that STE structures are unused, the SMMU_(S_)GBPA.IMPDEF field might be used. Otherwise, the STE.S2VMID field might be used. Details of these use cases are outside of the scope of this specification.

SMMU support for TLB maintenance messages that are broadcast from the PE is optional, but ARM recommends that support is implemented. These messages convey TLB invalidations from certain TLBI instructions on PEs. The ARM architecture [4] requires invalidation broadcasts to affect other PEs or agents in the same Inner Shareable domain. Broadcast TLB invalidate messages convey one or more of an address, an ASID or a VMID as required for a given invalidation operation, the scope defined by a translation stage to be affected and the translation regime in which the TLB entries are tagged. Support for broadcast invalidation is indicated by SMMU_IDR0.BTM.

Where SMMU_IDR0.BTM == 1, setting SMMU_(S_)CR0.PTM == 1 causes the SMMU to ignore broadcast TLB invalidation operations for the Security state. Broadcast TLB invalidation messages that would invoke an illegal operation, such as an invalidation that applies to a stage or Security state that is not implemented in the SMMU, are silently ignored, with the exception that messages that have a combined effect must affect the implemented stages and ignore any unimplemented stage. When SMMU_IDR0.S2P == 0, the SMMU matches VMID 0 for incoming broadcast TLB invalidation messages.

Note: On PEs that do not implement EL2, or that are running with stage 2 disabled, ARM expects software to configure VTTBR or VTTBR_EL2.VMID to 0. This ensures that for broadcast TLBI operations that include a VMID the VMID is set to 0.

Note: If stage 1-only NS-EL1 translations coexist with stage 1 and stage 2 NS-EL1 translations, then this mixed configuration must use distinct VMIDs in each configuration to avoid the stage 1-only translations matching lookups that use stage 1 and stage 2 configurations. This is not an issue where stage 1-only configurations are only present in EL2, EL2-E2H or Secure translation regimes.

Note: ARM expects broadcast invalidation from PEs to be used where address spaces are shared with the SMMU and common translations are maintained, such as Shared Virtual Memory applications, or stage 2 translations that share a common stage 2 translation table between VMs and the SMMU. When an address space is not shared with PE processes, broadcast TLB invalidations from the PEs to the SMMU have no useful effects and might over-invalidate unrelated TLB entries. Non-shared address spaces arise when a custom

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

address space is set up for a particular device – for example, scatter-gather or DMA isolation use cases. ARM expects that both shared and non-shared stage 1 translations might be in use simultaneously.

Stage 1 CDs contain an ASET flag, that represents the shared or non-shared nature of the ASID and address space. The set of ASIDs for non-shared address spaces might opt out of broadcast invalidation.

Note: ARM expects that SMMU stage 2 address spaces are generally shared with their respective PE virtual machine stage 2 configuration. If a particular SMMU stage 2 is to opt-out, ARM recommends that a hypervisor configures the STE with a VMID that is not allocated for virtual machine use on the PEs.

For a stage 1 configuration with a StreamWorld that has ASIDs, when CD.ASET == 1, the address space and ASID are non-shared. TLB entries that belong to that StreamWorld are not required to be invalidated by the broadcast invalidation operations that match with an ASID. These operations are VA{L}ExIS and ASIDExIS in the appropriate translation regime. All other matching broadcast invalidations are required to affect these entries. Where CD.ASET == 0, the ASID is considered shared with PE processes. TLB entries that belong to that StreamWorld are required to be affected by all matching broadcast invalidates. The definition of matching is identical to that of ARMv8-A PE TLBs [4]. CD.ASET does not affect the invalidation of stage 2 translation information.

For translation lookup of non-global TLB entries and command-based invalidation purposes, ASID values with CD.ASET == 0 are considered equivalent to ASID values with CD.ASET == 1. CMD_TLBI_* commands invalidate all matching TLB entries regardless of their ASET value. CD.ASET affects translation lookup of global TLB entries. For information about global TLB entry matching, see section 3.17.1.

A stage 1 configuration in a translation regime that does not have ASIDs, that is where StreamWorld == EL2 or StreamWorld == EL3, ignores the ASID field and is permitted but not required to tag TLB entries using ASETs. An equivalent semantic applies, in that ASET == 0 entries are affected by broadcast invalidation and ASET == 1 entries are not required to be invalidated by certain operations. EL2 TLB entries with ASET == 1 are not required to be invalidated by VA{L}ExIS or VAA{L}ExIS but must be invalidated by ALLE2IS. EL3 TLB entries with ASET == 1 are not required to be invalidated by VA{L}E3IS but must be invalidated by ALLE3IS.

Note: ARM does not anticipate that ASET == 1 has an effect on EL2 and EL3 contexts, however the behavior described here is consistent with other StreamWorld configurations.

Note: A broadcast invalidation operation that originates from a PE in EL2-E2H mode is not required to invalidate SMMU TLB entries that were inserted with StreamWorld == EL2, see section 3.17.5.

Note: See section 16.7.7 for AMBA interconnect DVM behavior with respect to ASET == 1.

Note: The ASID namespace might be affected by ASID rollover on the PE. These situations might be handled by:

- Refreshing the ASID namespace on the PE side and reallocating free ASIDs to new processes, but leaving ASIDs that are shared with SMMU contexts untouched.
- Swapping the ASID that is used in a CD, so that the old ASID is removed from the SMMU, and future traffic uses a freshly-allocated ASID. This can be achieved with an overlap in which both the old ASID and new ASID are active as the old ASID is updated in the CDs. This is followed by invalidation commands to the affected CDs (causing the SMMU to use the new ASID), and a CMD_SYNC. These steps are followed by TLB invalidation commands and a CMD_SYNC to remove all usage of the old ASID. When the final CMD_SYNC has ensured that these commands are complete, the old ASID can be considered free and the system can re-use it for a different address space.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.17.1 The Global flag in the Translation Table Descriptor

For translation regimes that have an ASID, ARMv8-A [4], defines an nG flag in the Translation Table Descriptors (TTDs) that allows TTDs to be marked as either global or non-global. A translation that is performed for a Secure stream is treated as non-global, regardless of the value of TTD.nG, if the descriptor is fetched from Non-secure memory. The EL2 and EL3 StreamWorlds do not support ASIDs, and the TTD.nG flag has no effect on these regimes.

When entries are global, an ASID is not required to be recorded in any resulting TLB entry. During lookup, a TLB entry marked as global can match regardless of the ASID that is provided with the lookup.

The TTD.nG flag does not allow a TLB entry to match a lookup from a StreamWorld that is not the same as the one from which the TLB entry was created.

Note: Global translations are used across address spaces with identical layout conventions, OS kernel addresses might be common across all process address spaces, and so they might be marked global. However, an SMMU might be used with many custom address spaces that are laid out in a manner convenient to the client device they serve, without common mappings.

The SMMU only matches global TLB entries, that is where TTD.nG == 0, against lookups from the same StreamWorld and ASID set (ASET). Global TLB entries with ASET == 0 do not match lookups through configurations with ASET == 1. Global TLB entries with ASET == 1 do not match lookups through configurations with ASET == 0.

Invalidation rules for non-locked global mappings are identical to those in ARMv8-A, TLBI ASIDE1IS and ASIDE2IS are not required to invalidate global mappings.

3.17.2 TLB maintenance from ARMv8-A PEs with EL3 in AArch64

When the Secure Stream table is controlled by an ARMv8-A PE where EL3 is using AArch64 state, software has the option of marking an STE as Secure or EL3 using the StreamWorld field, STE.STRW.

When the StreamWorld is Secure, the stream is configured on behalf of Secure-EL1 software and the resultant TLB entries are tagged as Secure, including an ASID if non-global. Such entries must be invalidated by:

- PE broadcast TLB invalidations (where supported and if CD.ASET allows) from Secure EL1 instructions. These instructions are:
  - TLBI VA{L}E1IS
  - TLBI VAA{L}E1IS
  - TLBI ASIDE1IS, for non-global entries
  - TLBALLE1IS
- SMMU invalidation commands on the Secure Command queue. These commands are:
  - CMD_TLBI_NH_ALL
  - CMD_TLBI_NH_ASID, for non-global entries
  - CMD_TLBI_NH_VAA
  - CMD_TLBI_NH_VA

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When StreamWorld is EL3, the stream is configured on behalf of EL3 software and resultant TLB entries are tagged as EL3, without ASID, which differentiates them from the Secure case above. Such entries are invalidated by:

- PE broadcast TLB invalidations (where supported and if CD.ASET allows) from EL3 instructions:
  TLBI VA{L}E3IS, TLBI ALLE3IS
- SMMU invalidation commands on the Secure Command queue:
  CMD_TLBI_EL3_VA, CMD_TLBI_EL3_ALL

3.17.3 TLB maintenance from ARMv7-A PEs or ARMv8-A PEs with EL3 using AArch32 state

When the Secure Stream table is controlled by an ARMv7-A PE or an ARMv8-A PE where EL3 is using AArch32 state, ARM expects software to mark the StreamWorld of an STE as Secure. The resultant TLB entries are tagged as Secure, including an ASID if non-global. Such entries are invalidated by:

- PE broadcast TLB invalidations (where supported and if CD.ASET allows) from Secure instructions. These instructions are:
  - TLBIMVA{L}IS
  - TLBIMVAA{L}IS
  - TLBIASIDIS, for non-global entries
  - TLBIALLIS
  - Note: TLBI VA{L}E3IS and TLBI ALLE3IS are AArch64-only and unavailable in this scenario.
- SMMU invalidation commands on the Secure Command queue. These commands are:
  - CMD_TLBI_NH_ALL
  - CMD_TLBI_NH_ASID, for non-global entries
  - CMD_TLBI_NH_VAA
  - CMD_TLBI_NH_VA

Note: Broadcast invalidations from ARMv7-A PEs or ARMv8 PEs where EL3 is using AArch32 might fail to invalidate SMMU TLB entries that are tagged with StreamWorld == EL3.

3.17.4 Broadcast TLB maintenance in mixed AArch32 and AArch64 systems and with mixed ASID or VMID sizes

Broadcast TLB maintenance instructions that are executed from ARMv7-A and ARMv8-A PEs using AArch32 and AArch64 Exception levels affect SMMU TLB entries (if broadcast TLB invalidation is supported and participation enabled), when the addresses, ASIDs, VMIDs and StreamWorlds match as appropriate.

The exceptions are:

- If a PE where EL3 uses AArch32 state issues an AArch32 TLBI that affects Secure entries, the TLBI is not required to affect SMMU TLB entries that were created with StreamWorld == EL3.
- If a PE where EL3 uses AArch64 state issues an AArch64 or AArch32 TLBI that affects Secure EL1 entries, the TLBI is not required to affect SMMU TLB entries created with StreamWorld == EL3.
- If a PE where EL3 uses AArch64 state issues an AArch64 TLBI that affects EL3 entries, the TLBI is not required to affect SMMU TLB entries created with StreamWorld == Secure.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

ARMv7-A PEs have 8-bit ASIDs and VMIDs. ARMv8-A PEs might have 16-bit ASIDs or VMIDs or both. An SMMU implementation supports 8-bit or 16-bit ASIDs and VMIDs, as indicated by SMMU_IDR0.{ASID16,VMID16}.

A difference in ASID or VMID size between the originator of the broadcast TLB maintenance instruction and the SMMU is resolved as follows. For each of ASID and VMID:

- An SMMU that supports a 16-bit ASID or VMID compares the incoming 16-bit broadcast value to its TLB tags directly and matches if the values are equal.
    - The incoming 16-bit value is constructed by the system from an originator with an 8-bit ASID or VMID by zero-extending the value to 16 bits.
- An SMMU that supports an 8-bit ASID or VMID compares the bottom 8 bits of the incoming broadcast values to its TLB tags:
    - The comparison is required to match if the bottom 8 bits are equal and the top 8 bits are zero.
    - The comparison is not required to, but might, match if the bottom 8 bits are equal but the top 8 bits are non-zero.

When the SMMU supports 16-bit ASIDs, that is when SMMU_IDR0.ASID16 == 1, it does so for all StreamWorlds that use ASIDs (NS-EL1, Secure, EL2-E2H). The SMMU does not differentiate ASID size by AArch32 contexts, as does an ARMv8-A PE and, if supported by an implementation, 16-bit ASIDs can be used in CDs where CD.AA64 == 0. ARM expects that legacy software will continue to write zero-extended 8-bit values in the ASID field in this case. The same behavior applies for 16-bit VMIDs, when SMMU_IDR0.VMID16 == 1, the behavior of which is not modified by STE.S2AA64 == 0.

3.17.5 EL2 ASIDs in EL2 Host (E2H) mode

The EL2 translation regime consists of:

- A stage 1 translation with one translation table without user permission checking.
- TLB entries that are not tagged with an ASID.

In E2H mode, the EL2 translation regime remains stage 1-only, but consists of two stage 1 translation tables that function the same way as those for an EL1 stage 1 translation. The resulting TLB entries are tagged as EL2, and might include an ASID. EL2-E2H mode can be configured for Non-secure STEs using STE.STRW when SMMU_IDR0.Hyp == 1 and SMMU_CR2.E2H == 1.

Note: In the ARMv8.1-A architecture [5], EL2-E2H mode is referred to as the Virtualization Host Extensions.

Broadcast TLB maintenance from a PE affects SMMU TLB entries when the whole system uses EL2 or EL2-E2H mode. Broadcast messages from a PE running in EL2-E2H supply an EL2 scope, and also supply an ASID to match. In addition, EL2-E2H mode extends the set of EL2 broadcast invalidations to the following operations:

- Invalidate all, for EL2-tagged entries.
- Invalidate by VA and ASID, for EL2.
- Invalidate by VA in all ASIDs, for EL2.
- Invalidate all by ASID, for EL2.

If a PE running at EL2 uses E2H mode, but an SMMU contains TLB entries that were inserted with StreamWorld == EL2 configuration, the EL2-E2H broadcast invalidations from the PE are not required to invalidate these TLB entries.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If a PE running at EL2 does not use E2H mode, but an SMMU contains TLB entries that were inserted with StreamWorld == EL2-E2H configurations, EL2 broadcast invalidations from the PE are not required to invalidate these TLB entries.

Note: If broadcast invalidation is required for translations controlled by EL2 software, ARM recommends that StreamWorld == EL2 is used when host PEs do not use E2H mode, and that SMMU_CR2.E2H == 1 and therefore StreamWorld == EL2-E2H are used when host PEs use the E2H mode.

An implementation is not required to differentiate TLB entries with StreamWorld == EL2 from those with StreamWorld == EL2-E2H. ARM expects that the SMMU is programmed in a way that ensures that TLB entries of both of these StreamWorlds never coexist in translation caches. Therefore:

- A change to SMMU_CR2.E2H must be accompanied by an invalidation of all TLB entries that were inserted with StreamWorld == EL2 or StreamWorld == EL2-E2H. See section 6.3.12 for details.
- The implementation of TLB invalidation commands CMD_TLBI_EL2_VAA and CMD_TLBI_EL2_VA might change depending on SMMU_CR2.E2H, see the individual commands for details.

Note: A TLB lookup through a configuration with StreamWorld == EL2-E2H, that is while SMMU_CR2.E2H == 1, matches the ASID of the configuration with the ASID tag of the EL2 TLB entry, unless the entry is marked global. A TLB insertion through the same configuration inserts a TLB entry tagged with the ASID of the configuration unless Global. When StreamWorld == EL2, that is while SMMU_CR2.E2H == 0, a TLB lookup does not match the ASID tag of EL2 TLB entries, nor does a TLB insertion tag the entry with a known ASID value. A change to SMMU_CR2.E2H can cause unexpected TLB entries to match.

3.17.6 VMID Wildcards

Some virtualization use cases involve the presentation of different views of page permissions in the same address space to different device streams. The mechanism by which one address space is split into more than one view is outside the scope of this document.

Note: STEs with different translation table pointers must have different VMIDs. TTBs in the same VMID are considered equivalent.

In the SMMU, SMMU_CR0.VMW controls a VMID wildcard function that enables groups of VMIDs to be associated with each other for the purposes of invalidation. An invalidation operation that matches on VMID matches one exact VMID or ignores a configured number of VMID LSBs, as configured by SMMU_CR0.VMW.

Note: For example, SMMU_CR0.VMW might configure 1 LSB of VMID to be ignored so an incoming broadcast invalidate for VMID `0x0020` matches TLB entries tagged with VMID `0x0020` or `0x0021`. This configuration allows VMIDs to be allocated in groups of adjacent or contiguous values, using one VMID in the PEs for the VM and one or more of the others to support different IPA address space views in different device stage 2 configurations.

Both broadcast TLB invalidation and explicit SMMU TLB invalidation commands, whether for stage 1 within the guest or at stage 2, affect all VMIDs that match the group wildcard when SMMU_CR0.VMW != 0.

Note: The broadcast TLB invalidation mechanisms that might exist on a PE and interconnect are not modified by this feature. The VMW field modifies the internal behavior of the SMMU on receipt of such a broadcast invalidation.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The VMID wildcard controlled by SMMU_CR0.VMW only affects a VMID that matches on invalidation. The SMMU continues to store all bits of VMID in the TLB entries that require them, and does not allow dissimilar VMID values to alias on lookup.

3.18 Interrupts and notifications

Events that are recorded to the Event queues, PRI requests and Global errors have associated interrupts to allow asynchronous notification to a PE.

An implementation might support *Message Signalled Interrupts* (MSIs) which take the form of a 32-bit data write of a configurable value to a configurable location, typically, in GICv3 systems, the GITS_TRANSLATER or GICD_SETSPI_NSR registers. For more information, see [7]. When SMMU_S_IDR1.SECURE_IMPL == 1, ARM expects notifications that were generated by Secure events to set Secure SPIs using the GICD_SETSPI_SR register in systems that implement the GICv3 architecture.

An implementation must support one of, or optionally both of, wired interrupts and MSIs. Whether an implementation supports MSIs is discoverable from SMMU_IDR0.MSI and SMMU_S_IDR0.MSI. An implementation might support wired interrupt outputs that are edge-triggered. The discovery of support for wired interrupts is IMPLEMENTATION DEFINED.

An implementation with support for two Security states, that is where SMMU_S_IDR1.SECURE_IMPL == 1, might implement MSIs for one Security state but not the other. ARM recommends that an implementation does not support Secure MSIs without also supporting Non-secure MSIs.

It is not permitted for an interrupt notification of the presence of new information to be observable before the new information is also observable. This applies to MSI and wired interrupts, when:

- A Global error condition arises. The change to the Global error register, GERROR, must be observable if the interrupt is observable.
- New entries are written to an output queue. The presence of the new entries must be observable to reads of the queue index registers if the interrupt is observable. See section 3.5.2.
- A CMD_SYNC completes. The consumption of the CMD_SYNC must be observable to reads of the queue index registers if the interrupt is observable.

Each MSI can be independently configured with a memory type and Shareability. This makes it possible to target a Device MSI target register or a location in Normal memory (that might be cached and shareable). See the SMMU_IDR0.COHACC field in section 6.3.1, which indicates whether the SMMU and the system support coherent accesses, including MSI writes.

Note: A PE might poll this location or might, for example in ARMv8-A PEs, wait for loss of an exclusive reservation that covers an address targeted by the notification. In this example, an ARMv8-A PE with an SMMU that is capable of making shared cacheable accesses can achieve the same behavior as a WFE wake-up event notification (see CMD_SYNC) without wired event signals, using MSIs that are directed at a shared memory location.

Note: If the destination of an MSI write is a register in another device, ARM recommends that it is configured with Device-nGnRnE or Device-nGnRE attributes.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The SMMU does not output inconsistent attributes as a result of misconfiguration. Outer Shareable is used as the effective Shareability when Device or Normal Inner Non-cacheable Outer Non-cacheable types are configured.

MSIs that are generated by Secure sources are performed with Secure accesses, that is NS == 0. MSIs from Non-secure sources are performed with Non-secure accesses, that is NS == 1. Apart from the memory type, Shareability and NS attributes of MSIs, all other attributes of the MSI write are IMPLEMENTATION DEFINED.

A GICv3 Interrupt Translation Service (ITS) differentiates interrupt sources using a DeviceID. To support this, the SMMU does the following:

a) Passes StreamIDs of incoming client device transactions. These generate DeviceIDs in a system-specific manner.

b) Produces a unique DeviceID of its own, one that does not overlap with those produced for client devices, for outgoing MSIs that originate from the SMMU. As with any other MSI-producing master, this is set statically in a system-defined manner.

SMMU MSIs are configured with several separate pieces of register state. The MSI destination address, data payload, Shareability, memory type and enables in combination construct the MSI write, onto which the unique DeviceID of the SMMU is attached.

Edge-triggered interrupts can be coalesced within the system interrupt controller. The SMMU can internally coalesce events and identical interrupts so that only the latest interrupt is sent, but any coalescence must not significantly delay the notification. This applies to both MSIs and edge-triggered wired interrupts.

When MSIs are not supported, the interrupt configuration register fields that would configure MSI address and data are unused. Only the interrupt Enable field is used.

3.18.1 MSI synchronization

The SMMU ensures that previously-issued MSI writes are completed at the following synchronization points:

- For register-based MSI configuration, the act of disabling an MSI through SMMU_(S_)IRQ_CTRL, see section 6.
- A CMD_SYNC ensures completion of MSIs that originate from the completion of prior CMD_SYNC commands that were consumed from the same Command queue.

Completion of an MSI guarantees that the MSI write is visible to its Shareability domain or, if an abort response was returned, ensures that the abort is visible in GERROR with the appropriate SMMU_(S_)GERROR.MSI_*_ABT_ERR flag.

Note: Completion of an MSI terminated with abort sets a GERROR flag but does not guarantee completion of a subsequent GERROR interrupt that might be raised to signal the setting of the flag.

The two synchronization points define a point in time, *t*, for the respective interrupt sources. If MSIs are related to occurrences before this point *t*, they do not become visible after point *t*.

In the case of register-based MSI configurations, the additional guarantee is made that MSIs triggered after the MSI is re-enabled will use the new configuration.

For more information on interrupt enable and synchronization, see 6.3.16.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.18.2 Interrupt sources

The SMMU has the following interrupt sources. Depending on the implementation, each interrupt source asserts a wired interrupt output that is unique to the source, or sends an MSI, or both.

Source	Trigger reason	Notes
Event queue	Event queue transitions from empty to non-empty	-
Secure Event queue		-
PRI queue	PRI queue interrupt condition, see section 6.3.36.	-
Command queue CMD_SYNC	Sync complete, with option of generated interrupt	MSI configuration (destination, data) present in command
Secure Command queue CMD_SYNC		
GERROR	Global Error activated in SMMU_GERROR registers	-
S_GERROR	Secure Global Error activated in SMMU_S_GERROR registers	-

Each interrupt source can be enabled individually through SMMU_IRQ_CTRL and SMMU_S_IRQ_CTRL if present. If enabled, a pulse is asserted on a unique wired interrupt output, if this is implemented. If enabled, an MSI is sent if MSIs are supported and if the MSI configuration of the source enables the sending of an MSI by using an ADDR value that is not zero.

This allows an implementation that supports both MSIs and wired interrupts to use both types concurrently. For example, the Secure programming interface might use wired interrupts (whose source would be enabled, but with the MSI ADDR == 0 to disable MSIs) and the Non-secure programming interface might use MSIs (whose source would be enabled and have MSI address and data configured).

The conditions that cause an interrupt to be triggered are all transient events and interrupt outputs are effectively edge-triggered. There is no facility to reset the pending state of the interrupt sources.

Where an implementation supports RAS features, additional interrupts might be present. The operation, configuration and assertion of these interrupts has no effect on any of the interrupts listed in this section for normal SMMU usage. See section 12 for more information on RAS features.

3.19 Power control

An implementation might support implementation specific automatic power saving techniques, for example, power and clock gating, retention states during idle periods of normal operation. All use of these techniques is functionally invisible to devices and software. If implemented, these automatic powerdown or retention states:

- Might retain all valid cache contents or might cause loss of cached information.
- Do not allow undefined cache contents to become valid, valid cache contents to change, or otherwise corrupt any SMMU state.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Seamlessly operate with wake on demand behavior in the event of incoming device transactions.

Alternatively, a system might allow an SMMU to be powered off at the request of system software, in an IMPLEMENTATION DEFINED manner. This state is requested when no further SMMU operation is required by the system. Software must not make accesses to the SMMU programming interface in this state. Where two Security states are supported, this state is not entered unless Non-secure and Secure drivers of an SMMU *both* request powerdown.

Because such a power off represents a complete loss of state and functionality, this state must only be used when all client devices and interconnect are quiescent. Software must disable client device DMA and ensure any SMMU commands, invalidations and transactions from client devices that are in progress are complete before requesting powerdown. If any existing transactions are in a stalled state at the time of the powerdown, they must be terminated with an abort. The behavior when a transaction arrives at the SMMU after the powerdown state is entered is UNPREDICTABLE.

On an IMPLEMENTATION DEFINED wakeup event, the SMMU must be reset and the return of the SMMU to software control is signaled through an IMPLEMENTATION DEFINED mechanism. The SMMU is then in a state consistent with a full reset and the SMMU registers are required to be re-initialized before client devices can be enabled.

3.19.1 Dormant state

Implementations might provide automatic powerdown modes during idle periods in which SMMU registers are accessible but internal structures might be powered down. An implementation might provide a hint to software, through the SMMU_STATUSR.DORMANT flag, that it contains no cached configuration or translation information, possibly because of cache powerdown. Software can use this flag to determine that no structure or TLB invalidation is required and avoid issuing maintenance commands.

When SMMU_STATUSR.DORMANT == 1, the SMMU guarantees that:

- No caches of any structures or translations are present.
- Any required configuration or translation information will access the information in the configuration structures or translation tables in memory.
- No pre-fetch of any configuration or translation data is in progress.
- If any structures or translations were altered in memory, no stale version will be used by the SMMU.

Software can make use of this flag by:

1. Altering translations or configuration structure data.
2. Testing the flag
    - If the flag is 0, issuing invalidation commands or broadcast invalidation messages to invalidate any potentially-cached copies.
    - If the flag is 1, avoiding invalidation of the altered structure.

An implementation is not required to support this hint, and software is not required to take note of this hint.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.20 TLB and configuration cache conflict

3.20.1 TLB conflict

A programming error might cause overlapping or otherwise conflicting TLB entries to be generated in the SMMU. When an incoming transaction matches more than one TLB entry, this is an error. An implementation is not required to detect any or all TLB conflict conditions, but ARM recommends that an implementation detects TLB conflict conditions wherever possible.

If an implementation detects a TLB conflict, all of the following apply:

- It aborts the transaction that caused the lookup that resulted in conflict.
- It attempts to record a F_TLB_CONFLICT event. The F_TLB_CONFLICT event contains IMPLEMENTATION DEFINED fields that might include diagnostic information that exposes implementation specific TLB layout.

If an implementation does not detect a TLB conflict experienced by a transaction, behavior is UNPREDICTABLE, with the restriction that a transaction cannot access a physical address to which the configuration of a stream does not explicitly grant access.

A TLB conflict never enables transactions to do any of the following:

- Match a TLB entry tagged with a different VMID to that under which the lookup is performed.
- Match a TLB entry tagged with a different Security state to that under which the lookup is performed.
- Match a TLB entry tagged with a different StreamWorld to that under which the lookup is performed.
- When stage 2 is enabled, access any physical address outside of the set of PAs configured in the stage 2 translation tables that a given transaction is configured to use.

Any failure to invalidate the TLB by code running at a particular level of privilege does not give rise to the possibility of a device under control of that level of privilege accessing regions of memory with permissions or attributes that could not be achieved at that same level of privilege.

Note: For example, a stream configured with StreamWorld == NS-EL1 must never be able to access addresses using TLB entries tagged with a different VMID, or tagged as EL2, EL3, or Secure.

A TLB conflict caused by a transaction from one stream must not cause traffic for different streams with other VMID, StreamWorld, or Security configurations to be terminated. ARM recommends that an implementation does not cause a TLB conflict to affect traffic for other ASIDs within the same VMID configuration.

3.20.2 Configuration cache conflicts

All configuration structures match a fixed-size lookup span of one entry with the exception of the STE, which contains a CONT field allowing a contiguous span of STEs to be represented by one cache entry.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

A programming error might cause an STE to be cached with a span that covers an existing cached STE, which results in an STE lookup matching more than one STE.

An implementation is not required to detect any or all configuration cache conflict conditions but ARM recommends that an implementation detects conflict conditions wherever possible.

If an implementation detects a configuration cache conflict, all of the following apply:

- The transaction that caused the lookup that resulted in conflict is aborted.
- The SMMU attempts to record a F_CFG_CONFLICT event. The F_CFG_CONFLICT event contains IMPLEMENTATION DEFINED fields that might include diagnostic information that exposes implementation-specific cache layout.

If an implementation does not detect a conflict experienced by a transaction, behavior is UNPREDICTABLE.

A configuration cache conflict cannot cause an STE to be treated as though it is associated with a different Security state.

3.21 Structure access rules and update procedures

3.21.1 Translation tables and TLB invalidation completion behavior

Translation table walks, caching, TLB invalidation, and invalidation completion semantics match those of ARMv8-A [4], including rules on prefetch and caching of valid translations only. If intermediate translation table data is cached in the SMMU (a walk cache) this is invalidated during appropriate TLB maintenance operations in the same way as it would be on a PE.

Explicit TLB invalidation maintains TLB entries when the translation configuration has changed, to ensure visibility of the new configuration. Translation configuration is the collective term for translation table descriptors and the set of SMMU configuration information that is permitted to be cached in the TLB. This maintenance is performed from a PE using TLBI broadcast invalidation or using explicit CMD_TLBI_* commands.

A broadcast TLB invalidation operation becomes visible to the SMMU after an Inner Shareable TLBI instruction is executed by a PE. A command TLB invalidation operation becomes visible to the SMMU after it is consumed from the Command queue.

A TLB invalidation operation is complete after all of the following become true:

- All TLB entries targeted by the scope of the invalidation have been invalidated.
- Any relevant HTTUs are globally visible to their Shareability domain as set out in section 3.13.4.
- No accesses can become visible to their Shareability domain using addresses or attributes that are not described by the translation configuration, as observed after the invalidation operation became visible. This means that invalidation completes after:
    - All translation table walks that could, prior to the start of the invalidation, have formed TLB entries that were targeted by the invalidation are complete, so that all accesses to any fetched levels of the translation table are globally visible to their Shareability domain. This applies to a translation table walk performed for any reason, including:
        - A translation table walk that makes use of walk caches that are targeted by the invalidation.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

        - A stage 2 translation table walk that are performed because of a stage 1 TTD fetch, CD fetch or L1CD fetch.

          Note: To achieve this, a translation table walk might be stopped early and the partial result discarded.
    - SMMU-originated accesses that were translated using TLB entries that were targeted by the invalidation are globally visible to their Shareability domain. These accesses are stage 1 TTD accesses, CD fetches or L1CD fetches.
    - Where a stage 2 invalidation targets TLB entries that might have translated a stage 1 TTD access, the stage 1 TTD access is required to be globally visible by the time of the invalidation completion, but neither the overall stage 1 translation table walk or the operation that caused the stage 1 translation table walk are required to be globally visible. Otherwise, for stage 1 and stage 1and stage 2 scopes of invalidation, all client device transactions that were translated using any of the TLB entries that were targeted by the invalidation are globally visible to their Shareability domain.
    - The result of an ATOS operation cannot be based on addresses or attributes that are not described by translation configuration that could have been observed after the invalidation operation became visible.

Note: An in-progress translation table walk (performed for any reason, including prefetch) can be affected by a TLB invalidation, if the TLB invalidation could have invalidated a cached intermediate descriptor that was previously referenced as part of the walk. The completion of a TLB invalidation ensures that a translation table walk that could have been affected by the TLB invalidate is either:

- Fully complete by the time the TLB invalidation completes.
- Stopped and restarted from the beginning.

Note: This ensures that old or invalid pointers to translation sub-tables are never followed after a TLB invalidation (whether broadcast or CMD_TLBI_*) is complete. Completion of a TLB invalidation means the point at which a broadcast invalidation sync completion is returned to the system (for example, on AMBA interconnect, completion of a DVM Synch), or, for CMD_TLBI_* invalidations, the completion of a later CMD_SYNC command.

Note: The architecture states that where a translation table walk is affected by a TLB invalidation, one option is that the walk is stopped by the completion of the invalidation. An implementation must give this appearance, which means no observable side-effects of doing otherwise could ever be observed. However, after an invalidation completion that affects prior translation table reads made before the invalidation, an implementation is permitted to make further fetches of a translation table walk if, and only if, it is guaranteed that these reads have no effect on the SMMU or the rest of the system, are not made to addresses with read side-effects and will not affect the architectural behavior of the system.

Translation cache entries (pertinent to a Security state) are not inserted when SMMU_(S_)CR0.SMMUEN == 0.

3.21.2 Queues

The SMMU does not write to the Command queue. The SMMU writes to the PRI and Event queues. ARM expects the PRI queue and Event queue to be read but not modified by the agent controlling the SMMU. Writes to the Command queue do not require any SMMU action to ensure that the SMMU observes the values written, other than a write of the PROD register of the Command that causes the written command entries to be considered valid for SMMU consumption. If the SMMU internally caches Command queue entries, no other explicit maintenance of this cache is required. ARM expects that the SMMU is configured to read the queue from the required Shareability domain in at least an IO-coherent manner, or that both the SMMU and other entities make non-cached accesses to the queue so that cache maintenance operations are not required.

To issue commands to the SMMU, the agent submitting the commands:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

1. Determines (using PROD/CONS indexes) that there is space to insert commands.
2. Writes one or more commands to the appropriate location in the queue.
3. Performs a DSB operation to ensure observability of data written in step (2) before the update in step (4).
4. Updates the Command queue's PROD index register to publish the new commands to the SMMU.

Software is permitted to write any entry of the Command queue that is in an empty location between CONS and PROD indexes.

The SMMU might read and internally cache any command that is in a full location between PROD and CONS indexes If a command is cached, the cache is not required to be coherent with PE caches but if it is not coherent the following rules apply:

- When the SMMU stops processing commands because of a Command queue error, or when the queue is disabled, the SMMU invalidates all commands that it might have cached.
- A cached command must only be consumed one time and no stale cached value can be used instead of a new value when the queue location is later reused for a new command.

Note: The first rule means software can fix up or replace commands in the queue after an error, or while the queue is disabled, without performing any other synchronization other than re-starting command processing.

Software must not alter memory locations representing commands previously submitted to the queue until those commands have been consumed, as indicated by the CONS index, and must not assume that any alteration to a command in a full location will be observed by the SMMU.

Software must only write the CONS index of an output queue (Event queue or PRI queue) in a consistent manner, with the appropriate incrementing and wrapping, unless the queue is disabled. If this rule is broken, for example by writing the CONS index with a smaller value, or incorrectly-wrapped index, the queue contents are UNKNOWN.

Software must only write the PROD index of the Command queue in a consistent manner, with the appropriate incrementing and wrapping, unless the queue is disabled or a command error is active, see section 7.1. If this rule is broken, one of the following CONSTRAINED UNPREDICTABLE behaviors occurs:

- The SMMU executes one or more UNPREDICTABLE commands.
- The SMMU stops consuming commands from the Command queue until the queue is disabled and re-enabled.

3.21.3 Configuration structures and configuration invalidation completion

The entries of the configuration structures, the Stream table and Context descriptors, all contain fields that determine their validity. An SMMU might read any entry at any time, for any reason.

STEs and CDs contain a valid flag, V. A structure is considered valid only when the SMMU observes it contains V == 1 and no configuration inconsistency in its fields causes it to be considered ILLEGAL.

Some structures contain pointers to subsequent tables of structures (STE.S1ContextPtr, L1ST.L2Ptr and L1CD.L2Ptr). If a structure is invalid, the pointers within it are invalid. The SMMU does not follow invalid pointers, whether speculatively or in response to an incoming transaction.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

STEs in a linear Stream table and L1ST descriptors in a multi-level Stream table are located through the SMMU_(S_)STRTAB_BASE address. Entries in these tables are not fetched if SMMU_(S_)CR0.SMMUEN == 0, because the base pointer is not guaranteed to be valid. These base pointers must be valid when the corresponding SMMUEN == 1. Configuration cache entries associated with a Security state are not inserted when, for that Security state, SMMU_(S_)CR0.SMMUEN == 0.

Similarly, CDs or L1CDs are located through the STE.S1ContextPtr and L1CD.L2Ptr pointers. A CD must never be fetched or prefetched unless indicated from a valid STE, meaning that the STE S1ContextPtr is valid and therefore the STE enables stage 1.

Note: A particular area of memory is only considered to be an STE or CD because a valid pointer of a certain type points to it (the SMMU_(S_)STRTAB_BASE or L1STD.L2Ptr or STE.S1ContextPtr or L1CD.L2Ptr pointers respectively). A CD cannot be prefetched from an address that is not derived directly from the CD table configuration in a valid STE, as an area of memory is not a CD unless a valid STE or L1CD.L2Ptr points to it. Similarly, an L1CD is not actually an L1CD structure unless a valid STE points to it.

A structure is said to be *reachable* if a valid pointer is available to locate the structure. Depending on the structure type, the pointer might be a register base address or a pointer within a precursor structure (either in memory or cached). When SMMUEN == 0, no configuration structures are reachable. Otherwise:

- An STE is reachable if it is within the table given by the base and size indicated in the SMMU_(S_)STRTAB_BASE_* registers for a linear Stream table, or if it is within the 2nd-level table indicated by a valid L1STD base and span for a two-level Stream table.
- A L1ST descriptor in a two-level Stream table is reachable if it is within the first-level table indicated by the base and size set in the SMMU_(S_)STRTAB_BASE_* registers.
- A CD is reachable if it is within the table given by the base and size indicated by a valid stage 1 S1ContextPtr and S1CDMax of a valid STE for a linear CD table, or if it is within the 2nd-level table indicated by a valid L1CD base and span for a two-level CD table.

An implementation does not fetch an unreachable structure. Walk of the tree of configuration tables does not progress beyond any invalid structure.

An implementation is permitted to fetch or prefetch any reachable structure at any time, as long as the generated address lies within the bounds of the table containing the structure. An implementation is permitted to cache any successfully fetched or prefetched configuration structure, whether marked as valid or not, in its entirety or partially. That is:

- Any STE or L1STD within the STE table (given by the base, size and intermediate table spans if appropriate) can be fetched and cached.
- Any CD or L1CD within a CD table can be fetched and cached.

When fetching a structure in response to a transaction, an implementation might read and cache more data than the required structures, as long as the limits of the tables are respected.

Note: Any change to a structure must be followed by the appropriate structure invalidation command to the SMMU, even if the structure was initially marked invalid.

Note: An *unreachable* structure cannot be fetched, because there is no valid pointer to it. However, a structure might be cached if it was fetched while the structure was reachable, even if it is subsequently made unreachable. For example, a valid STE could remain cached and later used after the SMMU_(S_)STRTAB_BASE* registers are

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

altered. Software must perform configuration cache maintenance upon changing configuration that might make structures unreachable.

A structure that is not actually fetched, such as a CD/L1CD or STE/L1STD that experiences an external abort (F_CD_FETCH or F_STE_FETCH) or a CD/L1CD that fails stage 2 translation, does not cause knowledge of the failure to be cached. A future access of the structure must attempt to re-fetch the structure without requiring an explicit configuration structure invalidation command before retrying the operation that caused the initial structure fetch.

Note: For example, where stage 2 is configured to stall, to progress a transaction that causes a CD fetch that in turn causes a stage 2 Translation-related fault (an event with Stall == 1), it is sufficient to:

1. Resolve the cause of the translation fault, for example by writing a Translation table entry.
2. Issue a TLB invalidation operation, if required by the translation table alteration.
3. Issue a CMD_RESUME, giving the StreamID and STAG appropriate to the event record.

An implementation must not:

- Read any address outside of the configured range of any table.
    - Speculative access of reachable structures is permitted, but address speculation outside of configured structures is not permitted.
- Cache any structure under a different type to the table from which it was read. For example, it must not follow the pointer of an STE to a CD and cache that CD (or any adjacent CD in the CD table) as anything non-CD, for example a TTD.

Software must ensure it only configures tables that are wholly contained in Normal memory.

A configuration invalidation operation completes after all of the following become true:

- All configuration cache entries targeted by the invalidation have been invalidated.
- No accesses can become visible to their Shareability domain using addresses or attributes that could not result from the configuration structures as observed after the invalidation operation became visible. This means that invalidation completes after:
    - Any client device transactions that used configuration cache entries that were targeted by the invalidation are globally visible to their Shareability domain.
    - Any configuration structure walks that used configuration cache entries that were targeted by the invalidation are complete so that all accesses to any fetched levels of the structures are globally visible to their Shareability domain. This applies to a configuration structure walk performed for any reason, including a configuration structure walk performed because of a prefetch, command, incoming transaction, ATOS or Translation Request.

An in-progress configuration structure walk (performed for any reason, including prefetch) can be affected by a configuration invalidation command (CMD_CFGI_*) if a cached intermediate structure that was previously referenced as part of the walk could have been invalidated. The completion of a configuration invalidation command (as determined by the completion of a subsequent CMD_SYNC) ensures that any configuration structure walk that could be affected by the invalidate is either:

- Fully completed by the time the CMD_SYNC completes.
- Stopped and restarted from the beginning after the CMD_SYNC completes.

Note: This ensures that old or invalid pointers to subsequent configuration structures are never followed after an invalidation is complete. For example, when an SMMU has an STE pointing to a two-stage CD table and is prefetching a CD, then on reading the L1CD pointer, a CMD_CFGI_STE is processed that invalidates the STE

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

that located the L1CD table. If the STE is made invalid, the pointer to the CD table is no longer valid and the SMMU must not continue to fetch the second-level CD after acknowledging to software that it considers the STE invalid. Software is free to re-use the memory used for the CD tables after receiving this acknowledgement, so continuing the prefetch after this point risks loading now-unrelated data. The SMMU must abort the fetch and not read the second-level CD, or must read the second-level CD before signaling the CMD_CFGI_STE/CMD_SYNC as complete to software.

Note: Refer to the note in section 3.21.1 regarding observability of whether a translation table walk is stopped. For a configuration table walk that is stopped by an affected invalidation completion, an implementation is permitted to perform further fetches of a configuration structure walk after the completion, based on affected prior configuration structure reads that were made before the invalidation if, and only if, it is guaranteed that these reads have no effect on the SMMU or the rest of the system, are not made to addresses with read side-effects, and will thus not affect the architectural behavior of the system.

The size of single-copy atomic reads made by the SMMU is IMPLEMENTATION DEFINED but must be at least 64 bits. Any single field within an aligned 64-bit span of a structure can be altered without first making the structure invalid. For example, to change the ASID in a CD, the ASID field can be written directly, followed by CMD_CFGI_CD and CMD_SYNC. However, if there are two fields separated so that one single 64-bit write cannot atomically alter both at the same time, the structure cannot be modified in this way. Non-single copy atomic writes might be visible to the SMMU separately and an inconsistent state might be cached (in which one field update has been read but another missed). The structure must, in this case, be made invalid, modified, then made valid, using the procedures described in section 3.21.3.1.

Note: In some systems, 64-bit single-copy atomicity is only guaranteed to addresses backed by certain memories. If software requires such atomicity, it must locate SMMU configuration structures in these memories. For example, in LPAE ARMv7 systems, main memory is expected to be used to contain translation tables, and is therefore required to support 64-bit single-copy atomicity.

When a structure is fetched, the constituent 64-bit double-words of a structure are permitted to be accessed by the SMMU non-atomically with respect to the structure as a whole and in any temporal sequence (maintaining the relative address sequence of the read portions).

3.21.3.1 Configuration structure update procedure

Note: The SMMU is not required to observe the structure word that contains the V flag in a particular order with respect to the other data in the structure. This gives rise to a requirement for an additional invalidation when transitioning a structure from V == 0 to V == 1.

Because the SMMU can read any reachable structure at any time, and is not required to read the double-words of the structure in order, ARM recommends that the following procedure is used to initialize structures:

1. Structure starts invalid, having V == 0.
2. Fill in all fields, leaving V == 0, then perform a DSB operation to ensure written data is observable from the SMMU.
3. Issue a CMD_CFGI_<STRUCT>, as appropriate.
4. Issue a CMD_SYNC, and wait for completion.
5. Set V to 1, then perform a DSB operation to ensure write is observable by the SMMU.
6. Issue CMD_CFGI_<STRUCT>, as appropriate.
7. Optionally issue a CMD_SYNC, and wait for completion. This must be done if a subsequent software operation, such as enabling device DMA, depends on the SMMU using the new structure.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

To make a structure invalid, ARM recommends that this procedure is used:

1. Structure starts valid, having V == 1.
2. Set V == 0, then perform a DSB operation to ensure write is observable from the SMMU.
3. Issue a CMD_CFGI_<STRUCT>, as appropriate.
4. Issue a CMD_SYNC, and wait for completion.

If software modifies the structure while it is valid, it must not allow the structure to enter an invalid intermediate state.

Note: Because the rules in section 3.21.3 disallow prefetch of a structure that is not directly reachable using a valid pointer, structures might be fully initialized (including with V == 1) prior to a pointer to the structure becoming observable by the SMMU. For example, a stage 1 translation can be set up with this procedure:

1. Allocate memory for a CD, initialize all fields including setting CD.V to 1.
2. Select an STE, initialize all fields and point to the CD, but leave STE.V == 0.
3. Perform a DSB operation to ensure writes are observable from the SMMU.
4. Issue a CMD_CFGI_STE and a CMD_SYNC and wait for completion.
5. Set STE.V to 1, then perform a DSB operation to ensure write is observable from the SMMU.
6. Issue a CMD_CFGI_STE and CMD_SYNC and wait for completion.

Note: No CMD_CFGI_CD is required because it is impossible for the CD to have been prefetched in an invalid state. However, a CMD_CFGI_CD must be issued as part of a procedure that subsequently makes the CD invalid.

3.22 Destructive reads and directed cache prefetch transactions

Some interconnect architectures might support the following types of transaction input to the SMMU:

1. RCI: Read with clean and invalidate:
   - A read transaction containing a hint side effect of clean and invalidate.
   - Note: The AMBA AXI-E interface ReadOnceCleanInvalidate transaction is an example of this class of transaction.
2. DR: Destructive read:
   - A read transaction with a data-destructive side-effect that intentionally causes addressed cache lines to be invalidated, without writeback, even if they are dirty.
   - Note: The AMBA AXI-E interface ReadOnceMakeInvalid transaction is an example of this class of transaction.
3. W-DCP: Write with directed cache prefetch:
   - A write transaction containing a hint that changes the cache allocation in a part of the cache hierarchy that is not on the direct path to memory. This class of operation does not include those with data-destructive side-effects.
   - Note: The following AMBA AXI-E interface transactions are examples of this class of transaction: WriteUniquePtlStash, WriteUniqueFullStash.
4. NW-DCP: A directed cache prefetch without write data:
   - A transaction that is neither a read nor a write, but performs a cache prefetch in a similar way to a write with directed cache prefetch, without the written data.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Note: The following AMBA AXI-E interface transactions are examples of this class of transaction: StashOnceShared, StashOnceUnique.

The side-effects of these transactions are hints and are therefore distinct from, and treated differently to, Cache Maintenance Operations. See section 16.7.2.

In SMMUv3.0, the architecture does not support these transactions, which are unconditionally converted on output as specified by the interconnect architecture.

In SMMUv3.1, these transactions are permitted to pass into the system unmodified when the transaction bypasses all implemented stages of translation, see section 3.22.3 for permitted memory types. This happens when:

- SMMU_(S_)CR0.SMMUEN == 0 for the Security state of the stream:
  - These transactions are affected by SMMU_(S_)GBPA overrides in the same way as the implementation treats ordinary transactions.
- SMMU_(S_)CR0.SMMUEN == 1 for the Security state of the stream, but the valid STE of the stream has STE.Config == `0b100`.
- The valid STE for the transaction has STE.S1DSS == `0b01` and STE.Config == `0b101`, and the transaction is supplied without a SubstreamID,

When the output interconnect does not support these types of transaction, or when the conditions described in sections 3.22.1, 3.22.2 and 3.22.3 apply, these classes of transaction are downgraded with the following transformations:

Input transaction class	Output/downgraded transaction class
Read with clean and invalidate (RCI)	No downgrade, or downgrade to ordinary read transaction[(1)]
Destructive read (DR)	Non-destructive read: An ordinary read transaction, or a read with a clean and invalidate side-effect.[(2)]
Write with directed cache prefetch (W-DCP)	Ordinary write transaction
Directed cache prefetch without write data (NW-DCP)	No-op: Transaction completes successfully with no effect on the memory system.

(1) It is IMPLEMENTATION DEFINED whether an implementation downgrades a RCI into a read, or whether this transaction remains unchanged. An implementation might only downgrade the RCI into a read if the output interconnect supports read but not RCI transactions.

(2) It is IMPLEMENTATION DEFINED whether a downgrade of a destructive read to a non-destructive read chooses to downgrade to an ordinary read or a RCI.

The RCI, DR, and W-DCP transactions are read or write transactions with an additional hint. The NW-DCP transaction is a hint without data transfer. An implementation is permitted to downgrade the transaction as described in this section, for any reason. The data transfer portion of these transactions, if present, is not a hint, and is treated in the same way as an ordinary read or write.

Note: Unless the SMMU has been explicitly configured to do so, ARM recommends that the common behavior of an implementation is to avoid downgrading these transactions.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3.22.1 SMMUv3.1 control of transaction downgrade

An SMMUv3.1 implementation supporting these classes of transactions provides STE.{DRE,DCP} controls to permit these classes of transaction to pass into the system without transformation when one or more stages of translation are applied. This does not include the case where the only stage of translation is skipped because of the value of STE.S1DSS.

When these controls are disabled, the respective class of transactions is downgraded as described in the previous section:

Input transaction class	Requirement to be eligible to pass into the system without class downgrade
Read with clean and invalidate	No additional requirements
Destructive read	STE.DRE == 1. If STE.DRE == 0, downgraded into non-destructive read (read, or read with clean and invalidate).
Write with directed cache prefetch	STE.DCP == 1. If STE.DCP == 0, downgraded into ordinary write.
Directed cache prefetch without write data	STE.DCP == 1. If STE.DCP == 0, downgraded into no-op.

A read with clean and invalidate is non-destructive and is not required to be transformed into a different class of transaction by the SMMU. The SMMU evaluates permissions for this type of transaction the same way it does for an ordinary read, see section 3.22.3. A read with clean and invalidate might be transformed as required by the final memory type or Shareability.

If a transaction is enabled to progress without downgrade, it can only progress if the required translation table permissions are present, as described in the next section. If the required permissions are not present, the transaction might still be downgraded or cause a fault.

3.22.2 SMMUv3.1 permissions model

When one or more stages of translation are applied to these transactions, the interaction with the permissions determined from translations is shown below. The behaviors listed here assume that translation for the given address has progressed up to permission checking, and that no higher-priority fault, for example a Translation fault or an Access flag fault, has occurred.

Transaction type	Required permissions	Behavior if permissions not met
Read with clean and invalidate[1]	Identical to ordinary read: Requires Read or Execute permission, (depending on input InD and INSTCFG) at a privilege appropriate to PnU input and STE.PRIVCFG.	Identical to ordinary read.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Transaction type	Required permissions	Behavior if permissions not met
Destructive read	Requires Read or Execute permission (depending on input InD and INSTCFG), and Write, at a privilege appropriate to PnU input and STE.PRIVCFG.	If no Write permission, downgraded as above into a read or read with clean and invalidate.[2] If no Read/Execute permission (as appropriate), identical to ordinary read.
Write with directed cache prefetch	Identical to ordinary write: Requires Write permission at privilege appropriate to PnU input and STE.PRIVCFG. Always 'Data'.[3]	Identical to ordinary write.
Directed cache prefetch without write data	Read or Write or Execute at privilege appropriate to PnU input and STE.PRIVCFG, at each enabled stage of translation. It is implementation specific whether this permission is evaluated as the effective combination of permissions at all stages, or evaluated at each stage separately[5].	Prefetch does not occur.[4]

(1) This includes the case where a destructive read is downgraded to a read with clean and invalidate because STE.DRE == 0.

(2) Though a DR requires write permission to progress into the system as a DR, it does not cause a Permission fault for write.

(3) The SMMU treats all writes as Data regardless of InD input and STE.INSTCFG.

(4) An NW-DCP is not a write and, if HTTU of Dirty state is enabled, does not mark a page Dirty. If an NW-DCP has the required permissions at a given stage of translation and HTTU of Access flag is enabled for that stage, AF is updated. If required permissions are not met for an NW-DCP at a given stage of translation, the transaction does not progress into the system. However, if the translation conditions permit an AF update, a coincidental speculative update of AF might occur.

(5) This applies to the case where a non-overlapping set of permissions is available at stage 1 versus stage 2. For example, a stage 1 read-only translation with a stage 2 write-only translation.

See section 3.13.7 for information on the behavior of HTTU for Destructive Reads. HTTU for RCI and W-DCP behaves the same as for an ordinary read or write, respectively.

A directed cache prefetch without write data (NW-DCP) does not cause faults in the SMMU. Where the interconnect architecture requires a response to an NW-DCP, and where the SMMU terminates an NW-DCP, the SMMU does not cause abort responses to be returned.

If RCI or DR ultimately lead to a fault, they are recorded as reads (data or instruction, as appropriate to input InD/INSTCFG).

If W-DCP ultimately leads to a fault, it is recorded as a write.

If the RCI, DR and W-DCP transactions lead to a fault, they stall in the same way as an ordinary read or write transaction if the SMMU is configured for stalling fault behavior. Retry and termination behave the same as for an

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

ordinary read or write transaction. If these transactions are stalled and retried, they are retried as the same transaction type.

3.22.3 SMMUv3.1 memory types and Shareability

The interconnect architecture of an implementation might impose constraints on the memory type or Shareability that output DR, RCI, W-DCP and NW-DCP operations can take.

At the point of final output the SMMU downgrades these operations, as described in 3.22, if the operations are not valid for output with the determined output attribute.

This rule applies to all such operations in all translation and bypass configurations, including:

- Global bypass (attribute set from GBPA).
- STE bypass (the only stage of translation is skipped because of STE.Config == `0b100` or STE.S1DSS == 0b01 and STE.Config == `0b101`).
- Translation.

Note: On AMBA AXI-E interfaces, the W-DCP operations (WriteUniquePtlStash, WriteUniqueFullStash) are not permitted to be emitted with a Non-shareable or Sys Shareability. The NW-DCP operations (StashOnceShared, StashOnceUnique) are not permitted to be emitted with Sys Shareability. RCI and DR operations (ReadOnceCleanInvalid and ReadOnceMakeInvalid) are not permitted to be emitted with NSH or Sys Shareability.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4 Commands

This section describes the behavior of commands given to the SMMU through the Command queue.

4.1.1 Command opcodes

All entries in the Command queue are 16 bytes long. Each command begins with an 8-bit command opcode, defined as follows:

Command opcode	Command name
0x00	Reserved
0x01	CMD_PREFETCH_CONFIG
0x02	CMD_PREFETCH_ADDR
0x03	CMD_CFGI_STE
0x04	CMD_CFGI_STE_RANGE Note: CMD_CFGI_ALL is an alias for a specific encoding of this command.
0x05	CMD_CFGI_CD
0x06	CMD_CFGI_CD_ALL
0x07-0x0A	Reserved
0x10	CMD_TLBI_NH_ALL
0x11	CMD_TLBI_NH_ASID
0x12	CMD_TLBI_NH_VA
0x13	CMD_TLBI_NH_VAA
0x14-0x17	Reserved
0x18	CMD_TLBI_EL3_ALL
0x19	Reserved
0x1A	CMD_TLBI_EL3_VA
0x1B-0x1F	Reserved
0x20	CMD_TLBI_EL2_ALL
0x21	CMD_TLBI_EL2_ASID
0x22	CMD_TLBI_EL2_VA
0x23	CMD_TLBI_EL2_VAA
0x24-0x27	Reserved
0x28	CMD_TLBI_S12_VMALL

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Command opcode	Command name
0x29	Reserved
0x2A	CMD_TLBI_S2_IPA
0x2B-0x2F	Reserved
0x30	CMD_TLBI_NSNH_ALL
0x31-0x3F	Reserved
0x40	CMD_ATC_INV
0x41	CMD_PRI_RESP
0x42-0x43	Reserved
0x44	CMD_RESUME
0x45	CMD_STALL_TERM
0x46	CMD_SYNC
0x47-0x7F	Reserved
0x80-0x8F	IMPLEMENTATION DEFINED
0x90-0xFF	Reserved

4.1.2 Submitting commands to the Command queue

Commands are submitted to the SMMU by writing them to the Command queue then, after ensuring their visibility to the SMMU, updating the SMMU_(S_)CMDQ_PROD.WR index which notifies the SMMU that there are commands to process. The SMMU does not execute commands beyond the CMDQ_PROD.WR index and, when commands are able to be processed as described in this chapter, a write to SMMU_(S_)CMDQ_PROD.WR is all that is required from software to cause the SMMU to consider newly-produced commands. See section 3.21.2.

Commands are able to be processed from the Command queue when all of the following conditions are met:

- The Command queue CONS and PROD indexes indicate that the queue is not empty.
- The Command queue is enabled through SMMU_(S_)CR0.CMDQEN.
- No Command queue error is active for the given Command queue.

The SMMU processes commands in a timely manner until all commands are consumed, or a command queue error occurs, or the queue is disabled.

When SMMU_IDR0.SEV == 1, the SMMU triggers a WFE wake-up event when a Command queue becomes non-full and an agent external to the SMMU could have observed that the queue was previously full. This applies to the Non-secure Command queue and, if implemented, the Secure Command queue.

Note: ARM expects that an attempt to insert a command into the queue will first observe from SMMU_(S_)CMDQ_CONS whether the queue has space, and if it is full might then poll the

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

SMMU_(S_)CMDQ_CONS index in a loop until the queue becomes non-full. Such a loop can be throttled using the WFE instruction when the SMMU and system supports sending of WFE wake-up events.

The behaviors of some commands are dependent on SMMU register state. Register state must not be altered between such a command having been submitted to the Command queue and the command completion. If a register field is changed while a dependent command could be being processed, it is UNPREDICTABLE whether the command is interpreted under the new or old register field value.

4.1.3 Command errors

A Command queue CERROR_ILL error occurs when:

- A Reserved command opcode is encountered.
- A valid command opcode is used with invalid parameters, see the individual command descriptions,
- A valid command opcode is used and a Reserved or undefined field is optionally detected as non-zero, which results in the command being treated as malformed,

Some commands, where specified, are IGNORED in certain circumstances. If a command causes a CERROR_ILL this takes precedence over whether the command is IGNORED or not.

A command queue CERROR_ABT error occurs when:

- An external abort is encountered upon accessing memory.

The SMMU stops command consumption immediately upon the first occurrence of an error, so that the SMMU_(S_)CMDQ_CONS.RD index indicates the command that could not be correctly consumed, and the error code is reported. See section 7.1 for details on Command queue error reporting and recovery.

4.1.4 Consumption of commands from the Command queue

A command is Consumed when the value observed in the SMMU_(S_)CMDQ_CONS.RD index register passes beyond the location of the command in the queue. This means that:

- As defined by the normal circular queue semantics, the location has been read by the SMMU and the producer might later re-use the location for a different command.
- Where explicitly noted in a command description, certain side-effects or guarantees have occurred.
    - Where not noted, no conclusions can be drawn from the Consumption of a command.

The SMMU_(S_)CMDQ_CONS index can be polled to determine whether a specific command has been Consumed. See section 4.7 for a summary of Consumption behavior.

A CMD_SYNC command is provided as a mechanism to ensure completion of commands submitted to the same queue at a location before the CMD_SYNC, including side-effects. A CMD_SYNC is not required to be issued in order to start the processing of earlier commands.

Note: A CMD_SYNC is used where it is necessary to determine completion of prior commands, such as a TLB invalidation, but commands are able to complete without depending on a CMD_SYNC.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.1.5 Reserved fields

All non-specified fields in the commands are RES0. An implementation is permitted to check whether these fields are zero. An implementation does one of the following:

- Detects non-zero use of a reserved field as a malformed command, resulting in CERROR_ILL.
- Ignores the entirety of any reserved fields.

Some combinations or ranges of parameter values are defined in this section to be ILLEGAL and use of these values results in a CERROR_ILL command error.

4.1.6 Common fields

These fields are common to more than one command and have the following behavior:

- SubstreamID and Substream Valid (SSV)
  - SSV indicates whether a SubstreamID is provided.

    SSV 0: SubstreamID not supplied.

    1: SubstreamID supplied.
- When SMMU_S_IDR1.SECURE_IMPL == 1, SSec is used by commands on the Secure Command queue to indicate whether the given StreamID parameter is Secure or Non-secure, in a similar way to SEC_SID for an input transaction (see section 3.10.1):

  SSec 0: StreamID is Non-secure

  1: StreamID is Secure
  - Commands on the Non-secure Command queue must set SSec == 0, where present, and cannot affect Secure streams. A command on the Non-secure Command queue with SSec == 1 is ILLEGAL, regardless of whether security is supported, and raises CERROR_ILL.
- Virtual address fields are Address[63:12], with [11:0] taken as zero.
- Physical, or IPA address fields are Address[51:12], or [51:2] for the case of MSIAddr, with other bits taken as zero.
  - Note: Bits[41:48] of these fields are RES0 in SMMUv3.0.

4.1.7 Out-of-range parameters

Providing an out-of-range parameter to a command has one of the following CONSTRAINED UNPREDICTABLE behaviors:

- The command has no effect
- The command has an effect, taking an UNPREDICTABLE value for the parameter that is out-of-range.
  - Note: For example, an implementation might truncate an out-of-range StreamID parameter to another in-range StreamID which might then be affected by the command.

See section 3.16.1.2. Some implementations might not provide the ability to express out-of-range values in certain fields.

A StreamID parameter for a command on the Non-secure Command queue is out of range if the value exceeds the implemented Non-secure StreamID size, as reported by SMMU_IDR1.SIDSIZE. For a command on the Secure Command queue, a Non-secure StreamID is out of range if the value exceeds the Non-secure

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

SMMU_IDR1.SIDSIZE and a Secure StreamID is out of range if the value exceeds the Secure SMMU_S_IDR1.S_SIDSIZE.

A SubstreamID parameter is out of range if the value exceeds the implemented SubstreamID size, as reported by SMMU_IDR1.SSIDSIZE.

Address parameter ranges are described for each command type that takes an address parameter.

The allowed range of ASID and VMID parameters is covered in section 4.4.

4.2 Prefetch

Two forms of prefetch command are available which can prefetch the configuration or translation data associated with a stream.

Valid prefetch commands with out of range parameters do not generate command errors. A request to prefetch an address that is out of range with respect to the translation table configuration for the StreamID or SubstreamID is IGNORED and does not record any kind of fault or error.

An implementation is not required to check the range of an Address parameter, but if it does then the parameter is considered out of range if:

- When the stream configuration enables stage 1 translation, the parameter has bits at Address[VAS-1] and upwards that are not all equal in value. TBI is permitted but not required to apply to the parameter.
- When the stream configuration enables stage 2 translation, the parameter has bits at Address[IAS] and upwards that are not zero.

If either SMMU_IDR1.SSIDSIZE == 0 or STE.S1CDMax == 0, then:

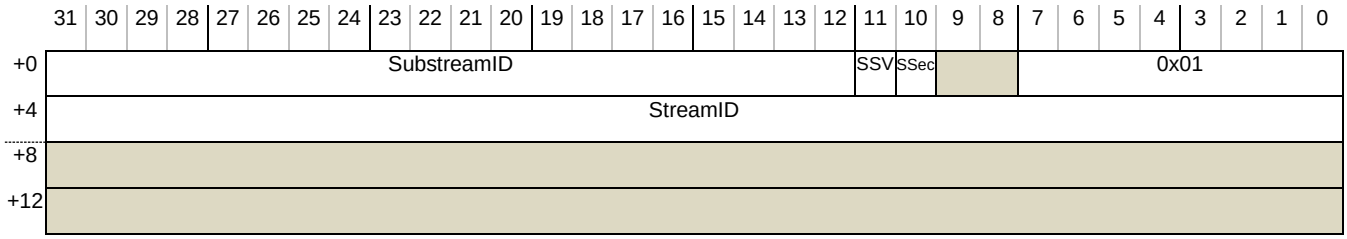
- Prefetch commands must be submitted with SSV == 0 and the SubstreamID parameter is IGNORED.
- Setting SSV == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:
    - The command behaves as though SSV == 0.
    - The command has no effect.

Prefetch commands directed to invalid configuration, for example, an STE or CD with V == 0 or out of range of the Stream table (or level-2 sub-table), fail silently and do not record error events. Similarly, prefetch commands directed to addresses that cause translation-related faults for any reason do not record error events.

When SMMU_(S_)CR0.SMMUEN == 0, valid prefetch commands are consumed but do not trigger a prefetch.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.2.1 CMD_PREFETCH_CONFIG(StreamID, SSec, SubstreamID, SSV)



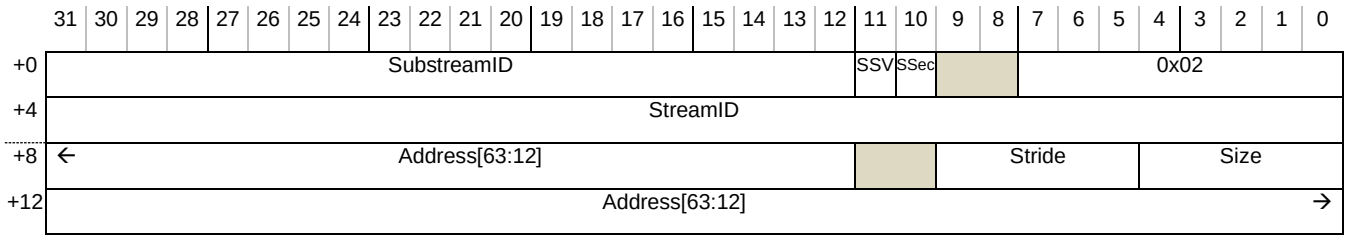
Prefetch any STE and CD configuration structures that are required to process traffic from the given StreamID, and SubstreamID if SSV == 1. An implementation is not required to prefetch any, or all, of the configuration requested.

If the STE of a StreamID configures both stage 1 and stage 2 translation, stage 2 HTTU is enabled, and if a CD or L1CD is fetched, then this command sets AF == 1 in the stage 2 TTD that is used to fetch the CD or L1CD.

Note: The CD might not be fetched if it is already cached.

When issued from the Secure Command queue, SSec controls whether StreamID is Secure or Non-secure. When issued from the Non-secure Command queue, StreamID is Non-secure and SSec must be zero. SSec == 1 is ILLEGAL on the Non-secure Command queue and results in CERROR_ILL.

4.2.2 CMD_PREFETCH_ADDR(StreamID, SSec, SubstreamID, SSV, Addr, Size, Stride)



Prefetch any STE and CD configuration structures and TLB entries for the given span of addresses as though accessed by transactions associated with StreamID, and SubstreamID if SSV == 1. An implementation is not required to prefetch any, or all, entries requested.

This command performs a prefetch of one or more TLB entries associated with a sequence of addresses given by:

$$\texttt{Addr} + (n * 2^{12+\texttt{Stride}}) \text{ where } 0 \le n < 2^{\texttt{Size}}.$$

The Stride parameter expresses the expected size of each resulting TLB entry, for the intended span. It controls the gap between successive addresses for which translations are prefetched. This parameter is encoded so that prefetches occur at a stride of $2^{12+\texttt{Stride}}$ bytes. For SMMUv3.0 implementations, Stride is RES0, must be set to 0, and the effective prefetch stride is 4KB. Use of a non-zero value either ignores the value or results in a CERROR_ILL.

The Size parameter expresses the desired number of prefetched translations, made for addresses at the effective Stride size, encoded as a 2^{Size} multiple of the stride size.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

An implementation internally limits the number of translation operations performed so that the overall prefetch operation completes in a reasonable time.

Note: An implementation might achieve this by ceasing prefetch at a point after which further prefetch would overwrite TLB entries prefetched earlier in the same operation.

Note: If translation table entries have been created for a range of addresses with a consistent page or block size, a prefetch operation can be optimized by setting Stride to align with the page or block size of the range.

Note: In some configurations, particularly those with smaller page sizes, it might negatively impact performance to request a prefetch of a span that results in insertion of a large number of TLB entries.

When HTTU is enabled, this command:

- Marks a stage 2 TTD as accessed for a CD fetch through stage 2 as described in 4.2.1 above.
- Performs HTTU in a manner consistent with that of a speculative read. See sections 3.14 and 3.13.

When issued from the Secure Command queue, SSec controls whether StreamID is Secure or Non-secure. When issued from the Non-secure Command queue, StreamID is Non-secure and setting SSec == 1 is ILLEGAL and results in CERROR_ILL.

4.3 Configuration structure invalidation

After an SMMU configuration structure is altered in any way, an invalidation command must be issued to ensure that any cached copies of stale configuration are discarded. The following commands allow invalidation of L1STDs, Stream table entries, L1CDs and CDs by StreamID and by StreamID and SubstreamID. All configuration structures must be considered to be individually cached and the agent controlling the SMMU cannot assume that invalidation of one type of structure affects those of another type unless explicitly specified, regardless of the implementation properties of a particular SMMU. See section 16.2.

Modifications of translation tables require separate invalidation of SMMU TLBs, using broadcast TLB invalidation or explicit TLB invalidation commands. See section 4.4.

Where a StreamID parameter is provided, it corresponds directly with an STE or L1STD. The StreamID parameter indicates that an STE is to be invalidated, or a CD that has been located directly via the indicated STE. The SSec parameter indicates whether the invalidation applies to configuration related to the Secure or Non-secure Stream table.

A structure invalidation command, at a minimum, invalidates all cached copies of structures directly indicated by the command parameters. The commands are permitted to over-invalidate by invalidating other entries.

Note: ARM recommends that implementations limit over-invalidation to avoid a negative performance impact.

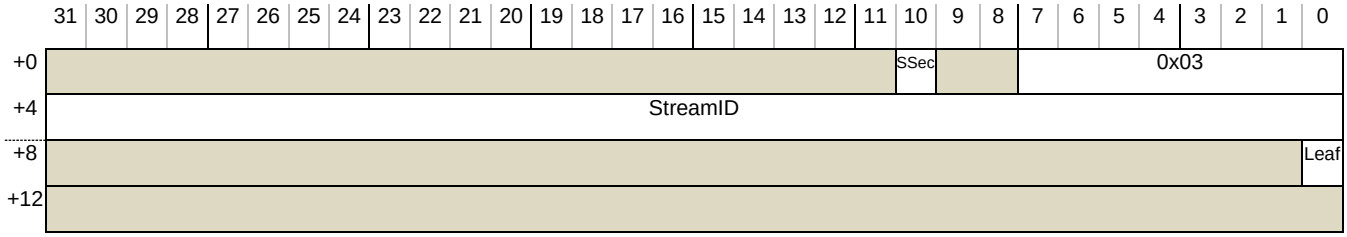
When issued from the Secure Command queue, a command might indicate a Secure or Non-secure Stream table entry and associated CD, using SSec. If over-invalidation occurs, it is permitted to affect either Security state.

Over-invalidation is permitted for non-locked configuration cache entries, but when issued from the Non-secure Command queue, ARM strongly recommends that a command only causes invalidation of cached copies of structures associated with Non-secure streams. Where IMPLEMENTATION DEFINED configuration cache locking is used, the IMPLEMENTATION DEFINED configuration cache invalidation semantics might restrict the effects of over-invalidation on locked configuration cache entries.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: ARM recommends that implementations choosing to allow over-invalidation consider the impact of Non-secure software being able to invalidate structures for Secure streams.

4.3.1 CMD_CFGI_STE(StreamID, SSec, Leaf)



Invalidate the STE indicated by StreamID and SSec.

This might be used for:

- Stream became invalid or valid.
- Enabling ATS.
- Enabling PASIDs.
- Changing stage 1 between bypass and translate.

Note: This command is not required to affect TLB contents. Separate TLB invalidation must be performed to clean up TLB entries resulting from a prior configuration.

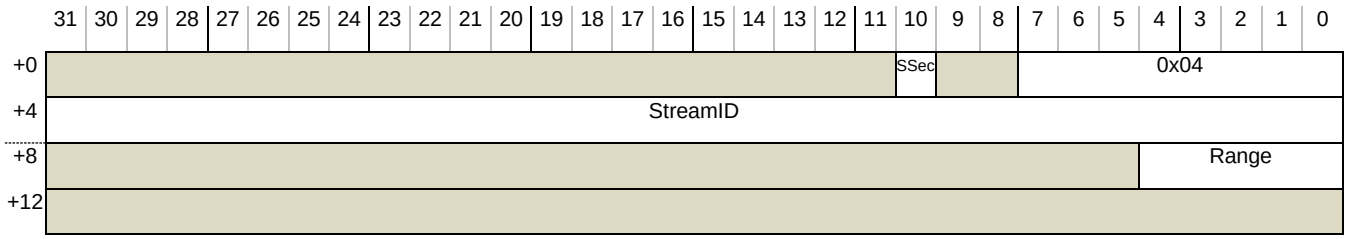
When Leaf == 0, this command invalidates all caching of the intermediate L1ST descriptor structures walked to locate the specified STE (as might be cached when multi-level Stream tables are used). When Leaf == 1, only the STE is invalidated and the intermediate L1ST descriptors are not required to be invalidated. An implementation is permitted to always invalidate the intermediate L1ST descriptors. STEs cached from linear Stream tables are invalidated with any value of Leaf.

This command invalidates all Context descriptors (including L1CD) that were cached using the given StreamID.

ARM recommends the use of the Leaf == 1 form of this command unless Leaf == 0 behavior is explicitly required. When a linear (single-level) Stream table is in use, the extra scope of the Leaf == 0 form is not required to be used.

Note: By avoiding Leaf == 0 invalidations unless cached intermediate pointers might exist from multi-level walks, invalidations might be faster and more power-efficient, depending on the implementation of STE caching.

4.3.2 CMD_CFGI_STE_RANGE(StreamID, SSec, Range)



Invalidate more than one STE, falling into the range of StreamIDs given by (inclusive):

Start = (StreamID & ~($2^{Range+1}$ - 1);

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

End = Start + $2^{(Range+1)}$ - 1;

Invalidation is performed for an aligned range of $2^{(Range+1)}$ StreamIDs. The Range parameter encodes a value 0-31 corresponding to a range of 2^1 - 2^{32} StreamIDs. The bottom Range+1 bits of the StreamID parameter are IGNORED, aligning the range to its size.

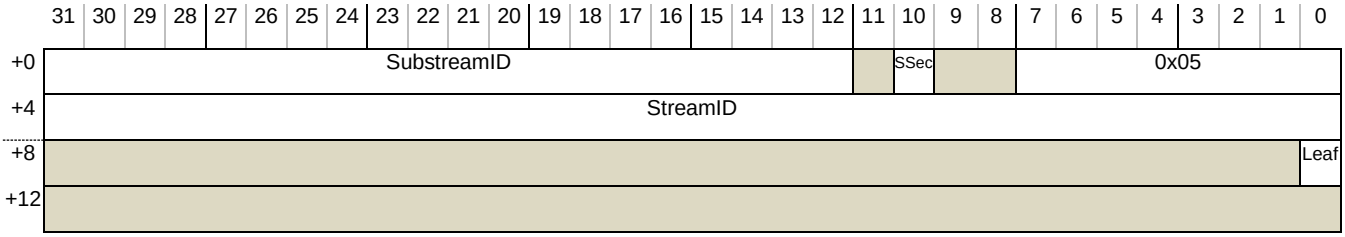
Note: ARM expects this command to be used for mass-invalidation when large sections of the Stream table are updated at the same time.

This command invalidates all caching of intermediate L1ST descriptors walked to locate the STEs in the given range (as might be cached when multi-level Stream tables are used). An implementation is permitted to over-invalidate these L1ST descriptors if required.

This command invalidates any Context descriptors (including L1CD) that were cached using all StreamIDs in the given range.

Note: CMD_CFGI_STE_RANGE(n, S, 31) invalidates all 2^{32} STEs, L1ST descriptors, CDs and L1CDs associated with the Security state given by S where S == 0 is Non-secure and S == 1 is Secure. This encoding is used for CMD_CFGI_ALL. The value of n is unimportant, as when Range == 31, all bits of StreamID are IGNORED.

4.3.3 CMD_CFGI_CD(StreamID, SSec, SubstreamID, Leaf)



Invalidate one CD.

This might be used when:

- Changing TTBRx/ASID (software must also invalidate TLBs using the old ASID).
- Enabling TBI.

The SubstreamID parameter indicates the Context descriptor to be invalidated. When a cached Context descriptor was fetched from index x of the CD table indicated by StreamID, then it is invalidated by this command when issued with SubstreamID == x. This includes the case where the STE indicates one CD which is equivalent to a CD table with one entry at index 0.

Note:

- Where substreams have been used with StreamID, that is when the STE located a CD table with multiple entries, the SubstreamID parameter indicates the CD to be invalidated as an index into the CD table.
    - STE.S1DSS might alter the translation behavior of the CD at index 0 (which might be used with SubstreamID 0, or transactions without a SubstreamID) but when issued with SubstreamID == 0, this command invalidates caching of the CD read from index 0 independent of its role in translation through S1DSS.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Where substreams are not used with the given StreamID, this command invalidates the CD when it is issued with SubstreamID == 0.

When the SubstreamID parameter is outside of the range of implemented SubstreamIDs, including the case where SMMU_IDR1.SSIDSIZE == 0 and the SubstreamID parameter is greater than 0, the behavior is consistent with the out-of-range parameter CONSTRAINED UNPREDICTABLE behavior described in section 4.1.7.

Note: An out-of-range SubstreamID parameter might cause this command to have no effect, or to operate on a different SubstreamID. In the case that SMMU_IDR1.SSIDSIZE == 0, a non-zero SubstreamID parameter might invalidate the single cached CD or have no effect.

Note: In the case where STE.S1DSS == `0b10`, non-SubstreamID traffic uses CD table entry 0, which would be invalidated using this command with the SubstreamID parameter equal to 0, although SubstreamID == 0 traffic is terminated (therefore is not associated with CD table entry 0), see STE.S1DSS for more information.

Note: The SubstreamID parameter is interpreted as a CD table index to invalidate. As such, a configuration with one CD can be thought of as a one-entry table. To invalidate the CD cached from this configuration, the SubstreamID parameter to this command would be 0.

When Leaf == 0, this command invalidates all caching of an intermediate L1CD descriptor that locates the CD in a 2-level CD table (see STE.S1Fmt). When Leaf == 1, intermediate L1CD descriptors are not required to be invalidated. An implementation is permitted to always invalidate the intermediate descriptors.

This command raises CERROR_ILL when stage 1 is not implemented.

A cached copy of CD data is treated as being local to the StreamID that locates the CD, because the CDs are indexed using SubstreamIDs whose scope is local to the StreamID. If multiple StreamIDs use a shared CD or table of CDs, the CD might be cached multiple times, having been fetched through any of the STEs. An invalidation command must be performed that affects all StreamIDs whose STEs point to the CD to be invalidated.

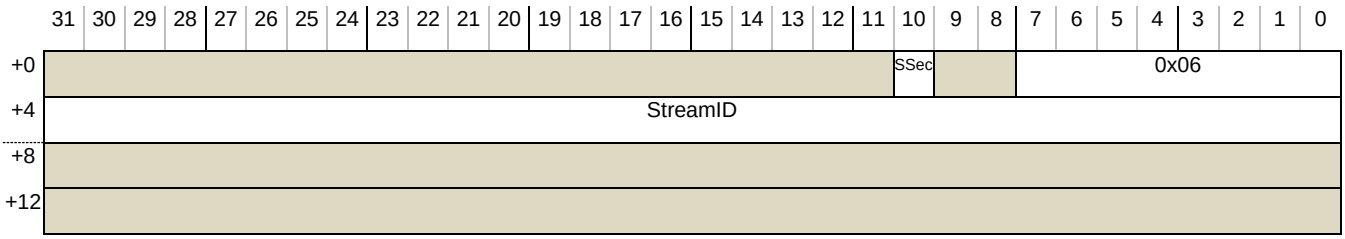
Note: This means that, when cached, CDs do not have to be located by address and might be indexed by StreamID and SubstreamID.

Note: For example, multiple CMD_CFGI_CD or CMD_CFGI_STE commands for each StreamID can be performed, or a wider-scope CMD_CFGI_STE_RANGE or CMD_CFGI_ALL covering all StreamIDs.

ARM recommends the use of the Leaf == 1 form of this command unless Leaf == 0 behavior is explicitly required. When a linear (single-level) CD Table is in use, the extra scope of the Leaf == 0 form is not required to be used.

Note: Avoiding Leaf == 0 invalidations unless cached intermediate pointers might exist from multi-level walks might have power and performance benefits.

4.3.4 CMD_CFGI_CD_ALL(StreamID, SSec)



Invalidate all CDs referenced by StreamID, for example when decommissioning a device. A separate command must also be issued to invalidate TLB entries for any ASIDs used, either by ASID or all.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

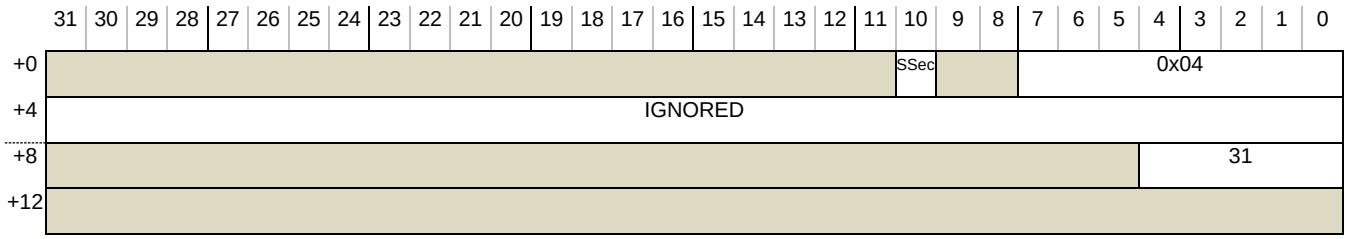
Note: When STE configuration has enabled substreams, this command affects all CDs cached for the StreamID substreams. When STE configuration has disabled substreams and used a single CD for stage 1, this command affects that single CD.

This command must also invalidate caches of all intermediate L1CD descriptors that locate CDs using the given StreamID.

This command raises CERROR_ILL when stage 1 is not implemented.

See 4.3.3, when a CD table is shared by multiple STEs this can give rise to multiple caches of each CD table entry. This command, or similar for STE or ALL scope, must be performed for all StreamIDs that could have cached the CD table contents.

4.3.5 CMD_CFGI_ALL(SSec)



This command is an **alias** for CMD_CFGI_STE_RANGE with Range == 31 (a range of 2^{32}, or all, StreamIDs). This command invalidates the cached configuration for all possible StreamIDs that are associated with the Security state given by SSec. Because Range == 31, the StreamID parameter is IGNORED.

Note: Use of SSec == 1 from the Non-secure Command queue raises CERROR_ILL, in the same way as for other commands.

Note: Behavior is as described in CMD_CFGI_STE_RANGE and invalidates caches of all configuration structures relevant to the Security state indicated by SSec. This includes caches of:

- Stream table entries.
- Intermediate or L1ST descriptor multi-level Stream table entries.
- Context descriptors.
- Intermediate or L1CD multi-level Context descriptor table entries.

Note: ARM recommends that an implementation explicitly detects this case and performs an invalidate-all operation instead of using an invalidate-range, if invalidate-all would be faster or more efficient.

Note: If it is required that TLB entries are also invalidated (such as reset-time initialization of the SMMU), ARM recommends that this command is *followed* by a command sequence to invalidate all TLB entries. A sequence in which TLB invalidations *precede* a CMD_CFGI_ALL might lead to a race in which TLB entries are pre-loaded using prefetch with possibly-stale cached configuration structures.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.3.6 Action of VM guest OS structure invalidations by hypervisor

Note: When a guest issues structure invalidation commands on its Command queue, the hypervisor must perform maintenance on its behalf. In particular, StreamIDs might need to be mapped from the guest view into real system StreamIDs. ARM recommends the following behavior:

Guest S1 command	Hypervisor action	Notes
CMD_CFGI_STE	Re-shadow STE, CMD_CFGI_STE	Map guest StreamID to host StreamID. Note: SubstreamID is the same in guest and host
CMD_CFGI_STE_RANGE	Re-shadow STEs, CMD_CFGI_STE or CMD_CFGI_STE_RANGE as appropriate.	
CMD_CFGI_CD	CMD_CFGI_CD	
CMD_CFGI_CD_ALL	CMD_CFGI_CD_ALL	

CMD_CFGI_ALL	CMD_CFGI_ALL, or,
	For each S in (GUEST_STREAMS) {
	CMD_CFGI_STE (S);
	}
		CMD_CFGI_ALL might over-invalidate and affect performance of other guests.
		
		An alternative is to explicitly invalidate structures for each StreamID assigned to the guest in question.

4.3.7 Configuration structure invalidation semantics/rules

Stalled transactions are unaffected by structure or TLB invalidation commands and must be dealt with either by using CMD_RESUME to retry or terminate them individually, or flushed using CMD_STALL_TERM for affected StreamIDs.

Note: When a stalled transaction is retried, it is re-translated as though it had just arrived, using newly-updated structures that might have been made visible to the SMMU with prior structure or TLB invalidation operations.

Translation of a transaction through the SMMU might not be a single atomic step and an invalidation command might be received while the transaction is in progress inside the SMMU. An invalidation of a structure that is used by a transaction that is in progress is not required to affect the transaction, if the transaction looked up the structure before it was invalidated. However, invalidation of any given structure must be seen as atomic so that a transaction must never see a partially-valid structure. A subsequent CMD_SYNC ensures that the transaction, having used a structure that was affected by an invalidation command, is visible to the system before the CMD_SYNC completes.

The consumption of structure and TLB invalidation commands does not guarantee invalidation completion. A subsequent CMD_SYNC is consumed when all prior invalidations of both structure and TLB have completed.

Refer to section 3.21 for structure update procedure and information about invalidation completion.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.4 TLB invalidation

The TLB invalidation commands are similar to the ARMv8-A broadcast TLB invalidation messages originating from PE TLB invalidation operations. These commands only affect the SMMU TLB and do not generate any broadcast operations to other agents in the system.

Note: SMMU TLB invalidate commands might be required because:

- The SMMU or interconnect does not support broadcast TLB invalidation messages.
- The ASET flag in a Context descriptor might cause a TLB entry to be marked as not participating in certain types of broadcast TLB invalidation, see section 3.17.

The ARMv8-A Last Level (leaf) scope is supported, to only invalidate an indicated TLB entry and last level cache.

Entries in this section show the PE TLB invalidation instructions that have the same scope as the SMMU command being described. Broadcast TLB invalidation messages from these PE operations trigger the equivalent operation on the SMMU. The scope of broadcast TLB invalidation and SMMU TLB invalidation commands are affected by VMID Wildcards, if enabled, see section 3.17.6 and SMMU_CR0.VMW.

ASID and VMID parameters to TLB invalidation commands are either 8-bit or 16-bit values, as appropriate, depending on whether the SMMU implementation supports 8-bit or 16-bit ASIDs and VMIDs (SMMU_IDR0.{ASID16,VMID16}). When support for either ASIDs or VMIDs is 8 bits, the upper 8 bits of the corresponding 16-bit parameter field are RES0. In this case, if the upper 8 bits are non-zero, the command is not required to affect TLB entries.

Commands matching TLB entries on ASID disregard the ASET value with which TLB entries were inserted.

Each command specifies the minimum required scope of TLB invalidation that must be performed. An implementation is permitted to invalidate more non-locked entries than this required scope (over-invalidation) but doing so can adversely affect performance and is not recommended by ARM. Where IMPLEMENTATION DEFINED TLB locking is used, invalidation semantics set out in section 16.6.1 must be observed.

Note: As described in section 4.3.7 above, a CMD_SYNC completes when prior TLB invalidations have completed. A sequence of CMD_TLBI_* commands followed by a CMD_SYNC is analogous to a sequence of TLBI instructions followed by a DSB on the PE, where the DSB ensures TLB invalidation completion.

Note: Consistent with the the ARMv8-A VMSA, for an entry to be eligible for invalidation, addresses that are provided for TLB invalidation are not required to be aligned to the start of a TLB entry address range. To match a TLB entry, the least significant bits of the address are ignored as needed, given the size of the entry.

4.4.1 TLB invalidation of stage 1

The following commands are available on a stage 1-only and a stage 1 and stage 2 SMMU. On a stage 2-only SMMU, they result in CERROR_ILL.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Stage 1 command	Stage 1 not supported	From Non-secure Command queue	From Secure Command queue, if present
CMD_TLBI_NH_ALL	CERROR_ILL	Invalidate NS EL1 stage 1 mappings	Invalidate Secure stage 1 mappings (inserted with StreamWorld == Secure). Note: EL3 entries are not required to be affected.
CMD_TLBI_NH_ASID			
CMD_TLBI_NH_VAA			
CMD_TLBI_NH_VA			
CMD_TLBI_EL3_ALL		CERROR_ILL	Invalidate EL3 stage 1 mappings (inserted with StreamWorld == EL3).
CMD_TLBI_EL3_VA			
CMD_TLBI_EL2_ALL		If SMMU_IDR0.Hyp == 1, invalidate EL2 or EL2-E2H stage 1 mappings as indicated by SMMU_CR2.E2H (see text). If SMMU_IDR0.Hyp == 0, CERROR_ILL.	
CMD_TLBI_EL2_VA			
CMD_TLBI_EL2_VAA			
CMD_TLBI_EL2_ASID			

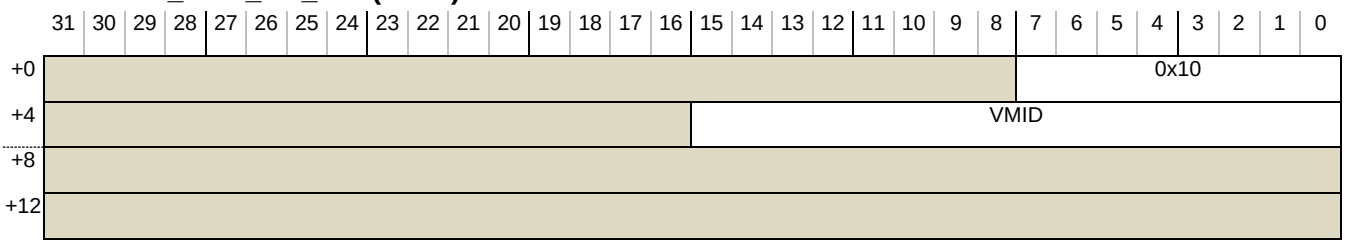
Note: ARM expects that software controlling a stage 1-only SMMU will use the first four commands. This includes driver software operating in a virtual machine controlling a stage 1-only SMMU interface.

In the following CMD_TLBI_NH_* commands, VMID is only matched when stage 2 is supported and the command is issued from the Non-secure Command queue. Otherwise, the VMID parameter is RES0 and, if it has a non-zero value, the SMMU is permitted to perform the invalidation on an UNKNOWN VMID value, or to not perform an invalidation.

Note: A stage 1-only implementation is not required to check that the VMID parameter of CMD_TLBI_NH_* is zero.

The Address parameters of these commands are VAs. An implementation is permitted but not required to treat the parameter as out of range if bits at Address[VAS-1] and upwards are not all equal in value. TBI is permitted but not required to apply to the parameter.

4.4.1.1 CMD_TLBI_NH_ALL(VMID)



When issued from the Non-secure Command queue, the invalidation scope is equivalent to that of TLBI VMALLE1{IS}, invalidates all stage 1 NS-EL1 (non-Hypervisor, non-EL2 or EL2-E2H) entries for VMID, including global entries.

When issued from the Secure Command queue, the invalidation scope is equivalent to that of a secure TLBI ALLE1{IS}, invalidates all Secure entries, including global entries.

For an equivalent to Non-secure TLBI ALLE1{IS}, see CMD_TLBI_NSNH_ALL.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.4.1.2 CMD_TLBI_NH_ASID(VMID, ASID)

	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
+0																									0x11							
+4	ASID																VMID															
+8																																
+12																																

The invalidation scope is equivalent to that of TLBI ASIDE1{IS}:

When issued from the Non-secure Command queue, invalidates stage 1 NS-EL1 non-global entries by ASID and VMID.

When issued from the Secure Command queue, invalidates stage 1 Secure non-global entries by ASID.

4.4.1.3 CMD_TLBI_NH_VAA(VMID, Addr, Leaf)

	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
+0																									0x13							
+4																	VMID															
+8	← Address[63:12]																														Leaf	
+12	Address[63:12]																															→

The invalidation scope is equivalent to that of TLBI VAA{L}E1{IS}:

When issued from the Non-secure Command queue, invalidates stage 1 NS-EL1 entries by VA for all ASIDs in VMID, including global entries.

When issued from the Secure Command queue, invalidates stage 1 Secure entries by VA for all ASIDs, including global entries.

When Leaf == 1, only cached entries for the last level of translation table walk are required to be invalidated.

4.4.1.4 CMD_TLBI_NH_VA(VMID, ASID, Addr, Leaf)

	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
+0																									0x12							
+4	ASID																VMID															
+8	← Address[63:12]																														Leaf	
+12	Address[63:12]																															→

The invalidation scope is equivalent to that of TLBI VA{L}E1{IS}:

When issued from the Non-secure Command queue, invalidates stage 1 NS-EL1 entries by VMID, ASID and VA, including global entries.

When issued from the Secure Command queue, invalidates stage 1 Secure entries by ASID and VA, including global entries.

When Leaf == 1, only cached entries for the last level of translation table walk are required to be invalidated.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.4.1.5 CMD_TLBI_EL3_ALL

	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
+0																									0x18							
+4																																
+8																																
+12																																

The invalidation scope is equivalent to that of TLBI ALLE3{IS}, invalidates all stage 1 EL3 entries.

This command is valid only on the Secure Command queue, otherwise a CERROR_ILL is raised.

4.4.1.6 CMD_TLBI_EL3_VA(Addr, Leaf)

	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
+0																									0x1a							
+4																																
+8	←												Address[63:12]																			Leaf
+12													Address[63:12]																			→

The invalidation scope is equivalent to that of TLBI VA{L}E3{IS}, invalidates stage 1 EL3 by VA. When Leaf == 1, only cached entries for the last level of translation table walk are required to be invalidated.

This command is valid only on the Secure Command queue, otherwise a CERROR_ILL is raised.

4.4.1.7 CMD_TLBI_EL2_ALL

	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
+0																									0x20							
+4																																
+8																																
+12																																

Invalidates all stage 1 EL2/Hyp entries, whether ASID-tagged (inserted in EL2-E2H mode when SMMU_CR2.E2H == 1) or not.

The invalidation scope is equivalent to that of TLBI ALLE2{IS}, and when HCR_EL2.TGE == 1 && HCR_EL2.E2H == 1, TLBI VMALLE1{IS}.

When SMMU_IDR0.Hyp == 0, this command causes a CERROR_ILL.

4.4.1.8 CMD_TLBI_EL2_VA(ASID, Addr, Leaf)

	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
+0																									0x22							
+4							ASID																									
+8	←												Address[63:12]																			Leaf
+12													Address[63:12]																			→

Invalidates stage 1 EL2/Hyp entries by VA, including global.

When Leaf == 1, only cached entries for the last level of translation table walk are required to be invalidated.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

This behavior of the command is governed by SMMU_CR2.E2H:

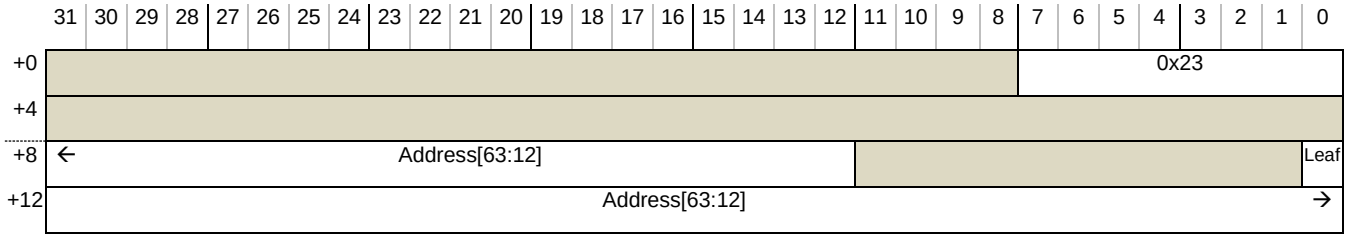
- When E2H == 1, TLB entries inserted with a StreamWorld == EL2-E2H configuration are invalidated if ASID matches (or global) and VA matches. TLB entries inserted with StreamWorld == EL2 are not required to be invalidated.
- When E2H == 0, TLB entries inserted with a StreamWorld == EL2 configuration are invalidated if VA matches, and the ASID parameter is ignored. TLB entries inserted with StreamWorld == EL2-E2H are not required to be invalidated.

This command must not be submitted to a Command queue when the effective value of E2H is UNKNOWN. Otherwise, it is UNPREDICTABLE whether any EL2 or EL2-E2H entries that match Addr and ASID are invalidated. See 6.3.12.3.

The invalidation scope is equivalent to that of TLBI VA{L}E2{IS}, (and when HCR_EL2.TGE == 1 && HCR_EL2.E2H == 1, TLBI VA{L}E1{IS.

When SMMU_IDR0.Hyp == 0, this command causes a CERROR_ILL.

4.4.1.9 CMD_TLBI_EL2_VAA(Addr, Leaf)



Invalidates stage 1 EL2/Hyp entries by VA for all ASIDs, including global.

When Leaf == 1, only cached entries for the last level of translation table walk are required to be invalidated.

This behavior of this command is governed by SMMU_CR2.E2H:

- When E2H == 1, TLB entries inserted with a StreamWorld == EL2-E2H configuration are invalidated if the VA matches, for all ASIDs. TLB entries inserted with StreamWorld == EL2 are not required to be invalidated.
- When E2H == 0, TLB entries inserted with a StreamWorld == EL2 configuration are invalidated if the VA matches. TLB entries inserted with StreamWorld == EL2-E2H are not required to be invalidated.

This command must not be submitted to a Command queue when the effective value of E2H is UNKNOWN. Otherwise, it is UNPREDICTABLE whether any EL2 or EL2-E2H entries that match Addr and ASID are invalidated. See section 6.3.12.3.

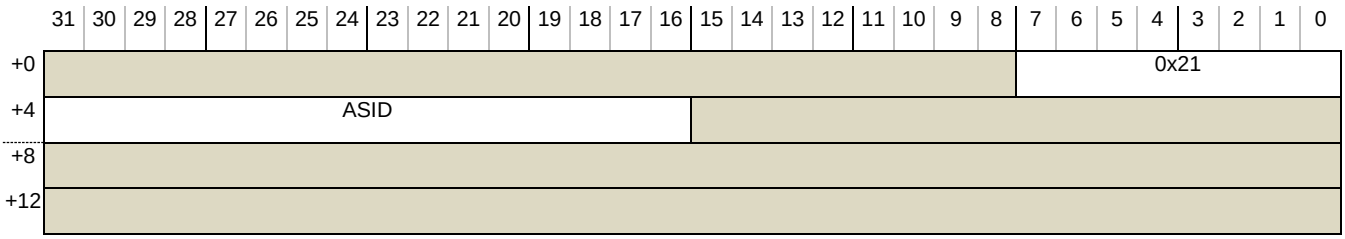
Note: An implementation might choose to optimise search of a TLB given this information, and access a single location for the given VA.

The invalidation scope is equivalent to that of TLBI VAA{L}E1{IS} when HCR_EL2.TGE == 1 && HCR_EL2.E2H == 1. When a PE is not in EL2-E2H mode, it does not have an equivalent.

When SMMU_IDR0.Hyp == 0, this command causes CERROR_ILL.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.4.1.10 CMD_TLBI_EL2_ASID(ASID)



Invalidates stage 1 EL2/Hyp non-global entries by ASID.

Non-global TLB entries inserted with a StreamWorld == EL2-E2H configuration are invalidated if ASID matches. EL2 (non-ASID-tagged) entries are not required to be invalidated.

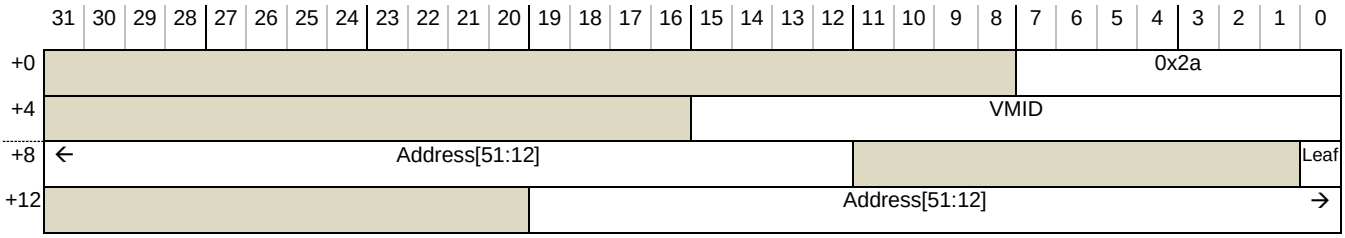
The invalidation scope is equivalent to that of TLBI ASIDE1{IS} when HCR_EL2.TGE == 1 && HCR_EL2.E2H == 1. When a PE is not in EL2-E2H mode, it does not have an equivalent.

When SMMU_IDR0.Hyp == 0, this command causes a CERROR_ILL.

4.4.2 TLB invalidation of stage 2

These commands allow a hypervisor to perform stage 2 invalidations on behalf of a VM, and are valid from both the Non-secure and Secure Command queue:

4.4.2.1 CMD_TLBI_S2_IPA(VMID, Addr, Leaf)



The invalidation scope is equivalent to that of TLBI IPAS2{L}E1{IS}, invalidates stage 2 by VMID and IPA.

When Leaf == 1, only cached entries for the last level of translation table walk are required to be invalidated.

When issued on an SMMU without stage 2 support, a CERROR_ILL is raised.

The Address parameter is an IPA. An implementation is permitted but not required to treat the parameter as out of range if bits at Address[IAS] and upwards are not all zero. Address bits [11:0] are treated as 0s.

In SMMUv3.0, Address bits [51:48] are RES0.

Consistent with ARMv8-A [4], this command is not required to invalidate structures containing combined stage 1 and stage 2 information (from nested stage 1 and 2 translation configuration). Where combined Stage 1 and stage 2 mappings are possible, this command is expected to be used with CMD_TLBI_NH_ALL or CMD_TLBI_NH_VAA. However, this command is sufficient to invalidate translations resulting from stage 2-only configuration.

Note: An implementation choosing to combine all stages of translation into one TLB must distinguish whether an entry represents a VA or an IPA (inserted from stage 1, stage 1 and 2 configuration or stage 2-only) so that invalidation by IPA is able to locate entries inserted from stage 2-only configurations.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.4.2.2 CMD_TLBI_S12_VMALL(VMID)

	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
+0																									0x28							
+4																	VMID															
+8																																
+12																																

The invalidation scope is equivalent to that of TLBI VMALLS12E1{IS}, invalidates all Non-secure (and non-Hyp) entries at all implemented stages for VMID. When issued on an SMMU without stage 2 support, a CERROR_ILL is raised.

4.4.3 Common TLB invalidation

This command is valid whether the SMMU supports only stage 1, only stage 2 or both stages:

4.4.3.1 CMD_TLBI_NSNH_ALL

	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
+0																									0x30							
+4																																
+8																																
+12																																

The invalidation scope is equivalent to that of Non-secure TLBI ALLE1{IS}, valid from both the Non-secure and Secure Command queue, this command invalidates all Non-secure, non-EL2, non-EL2-E2H TLB entries at all implemented stages.

Note: A full TLB invalidate would be achieved by:

- (S) CMD_TLBI_EL3_ALL (Clear EL3 stage 1)
- (S) CMD_TLBI_NH_ALL (Clear Secure stage 1)
- (NS) CMD_TLBI_EL2_ALL (Clear EL2/Hyp stage 1)
- (NS) CMD_TLBI_NSNH_ALL (Clear EL1/Non-secure stage 1 and stage 2)

On reset, ARM recommends that the Secure driver issues CMD_TLBI_EL3_ALL and CMD_TLBI_NH_ALL, and that the Non-secure driver issues CMD_TLBI_EL2_ALL and CMD_TLBI_NSNH_ALL. Non-secure software must not assume Secure software has invalidated Non-secure configuration caches or TLB contents.

Note: A stage 1-only Non-secure SMMU with SMMU_IDR0.Hyp == 0 can also be initialized with CMD_TLBI_NH_ALL.

4.5 ATS and PRI

These commands issue outgoing requests on the SMMU ATS port, to a connected Root Complex.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Software must not use these commands if ATS and PRI are not fully supported by all system components beyond the SMMU, such as a Root Complex or Endpoint, even if SMMU ID registers indicate support within the SMMU. An SMMU and system remain fully functional even if these commands are used without full system ATS and PRI support and do not deadlock. In this scenario these commands behave as a NOP.

Note: StreamID is considered Non-secure, because ATS and PRI are supported only for Non-secure streams.

4.5.1 CMD_ATC_INV(StreamID, SubstreamID, SSV, Global, Address, Size)

	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
+0	SubstreamID																				SSV		G		0x40							
+4	StreamID																															
+8	← Address[63:12]																								Size[5:0]							
+12	Address[63:12] →																															

Sends an invalidation request to StreamID, and SubstreamID when SSV == 1, for translations spanned by Address plus span of 4096 * 2^{Size}. The Address span is aligned to its size by the SMMU. The effective value of Address[11 + Size:0] is taken as zero.

All bits of Address[63:N] are conveyed to the endpoint, where N == 12 + Size.

A Size value of 52 corresponds to a 2^{64} byte span, meaning invalidate all. Use of values greater than 52 in an otherwise valid CMD_ATC_INV are permitted, but not required, to raise a CERROR_ILL, or might cause an UNKNOWN span to be used, which might mean no invalidation occurs.

In systems that do not support PASIDs, SubstreamID is IGNORED and SSV must be 0. If SSV == 1, one of the following CONSTRAINED UNPREDICTABLE behaviors occurs:

- The command behaves as though SSV == 0, issuing an invalidate to the given StreamID without a PASID TLP prefix.
- The command has no effect.

In systems that support PASIDs, a PCIe PASID TLP prefix containing SubstreamID is used on the ATC Invalidation Request message when SSV is set to 1. In addition, setting the Global (G) parameter to 1 when SSV == 1 sets the Global Invalidate flag in the Invalidation Request. The Global parameter is IGNORED when SSV == 0.

Note: In the PCIe ATS protocol, the Global Invalidate flag in the Invalidation Request message is reserved when no PASID prefix is used.

Note: When SSV == 1, SubstreamID is always used as the PASID in the PASID TLP prefix. Global has no effect on this property.

Note: If the SMMU is configured, through STE.S1DSS == `0b10`, to treat non-PASID (non-SubstreamID) Translation Requests as using the CD at index 0 of a table of CDs, the request is internally treated similar to a request with Substream 0 in an STE.S1DSS == `0b00` configuration, but as with the request, the response is given without PASID. An invalidation of a translation in this situation has SSV == 0.

If SMMU_IDR0.ATS == 0, a CERROR_ILL is raised. If SMMU_IDR0.ATS == 1 and SMMU_S_IDR3.SAMS == 1, and if this command is issued on the Secure Command queue, a CERROR_ILL is raised. If SMMU_IDR0.ATS == 1, but itATS is not supported by the rest of the system, this command behaves as a NOP. If ATS is supported but the SMMUEN that is associated with a given Command Queue is zero, this command behaves as a NOP.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

This command is permitted in either Non-secure or Secure Command queues, but the StreamID is always considered to be Non-secure.

The Consumption of CMD_ATC_INV does not guarantee invalidation completion. Completion is ensured by the completion of a subsequent CMD_SYNC. In a similar way to a TLBI operation, transactions that could have been translated using TLB entries targeted by the invalidation must all be visible to their Shareability domain before the ATC invalidation is considered complete.

Note: In a system with separate Root Complex and SMMU components, the protocol between the two must enforce this ordering, for example, the Root Complex signals that an ATC invalidation is complete only when affected transactions have been pushed to the SMMU and the SMMU then ensures this ordering is maintained, so that the completion signal is not observed before the affected transactions can be observed.

To invalidate a translation from an ATC, software must first invalidate SMMU caches of the translation (whether by explicit command or using broadcast invalidation) and then, after ensuring completion of the SMMU translation invalidate, issue a CMD_ATC_INV to invalidate ATC translations.

Note: For example, to alter and invalidate the last-level (leaf) translation table descriptor for a Non-secure EL1 translation of address VA using explicit SMMU operations:

1. Change translation table entry (and barrier to make visible to Shareability domain).
2. CMD_TLBI_NH_VA(VMID, ASID, VA, Leaf == 1).
3. CMD_SYNC.
4. CMD_ATC_INV(SID, SSID, SSV, Global == 0, VA).
5. CMD_SYNC (and wait for completion)
6. The new translation table entry at VA is guaranteed to be in use.

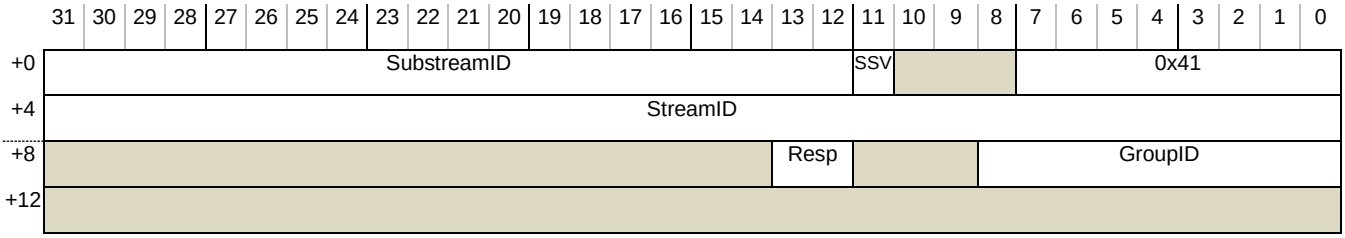
Note: System software is expected to associate a device with a translation context and translation table and, when changes to the translation table are made, perform SMMU TLB maintenance using the translation table's associated ASID/VMID and ATC maintenance of all devices associated with that translation table using the StreamID/SubstreamIDs of the devices.

Note: Software must not provide a Size parameter that is lower than the system STU, as invalidation is not guaranteed if this is done. If a request is made with a span smaller than the STU, the PCIe ATS specification [3] permits an endpoint to respond using a UR, or to round up the span and perform the invalidate with STU size.

Note: If this command results in a UR response from the endpoint, the command completes without error but invalidation is not guaranteed, see section 3.9.1.3.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.5.2 CMD_PRI_RESP(StreamID, SubstreamID, SSV, GroupID, Resp)



Notifies a device corresponding to StreamID, and SubstreamID when SSV == 1, that page request group GroupID has completed with given response, Resp.

GroupID: PRI GroupID from page requests

Resp: `0b00`: ResponseFailure: Permanent non-paging error (for example ATS or PRI is disabled for the device)

`0b01`: InvalidRequest: Page-in unsuccessful for one or more pages in the group

`0b10`: Success: Page request group was paged in successfully

`0b11`: Reserved ILLEGAL

In systems that do not support PASIDs, SubstreamID is IGNORED and SSV must be 0. If SSV == 1, one of the following CONSTRAINED UNPREDICTABLE behaviors occurs:

- The command behaves as though SSV == 0, issuing a response to the given StreamID without a PASID TLP prefix.
- The command has no effect.

In systems that support PASIDs, SSV == 1 results in a PCIe PASID TLP prefix.

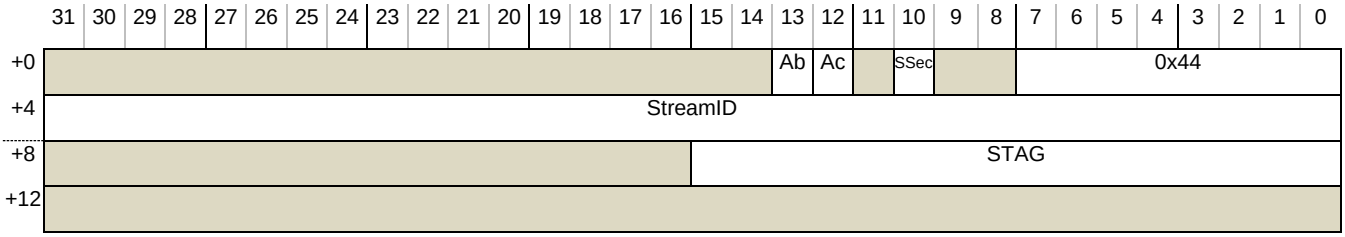
If PRI is not supported by an implementation (SMMU_IDR0.PRI == 0), or if a reserved parameter value is used, a CERROR_ILL is raised. If SMMU_IDR0.PRI == 1 and SMMU_S_IDR3.SAMS == 1, and if this command is issued on the Secure Command queue, a CERROR_ILL is raised. Otherwise, if PRI is supported by an SMMU but not supported by the rest of the system, this command behaves as a NOP. If PRI is supported but SMMUEN == 0, this command behaves as a NOP.

This command is permitted in either Non-secure or Secure Command queues, but StreamID is always considered to be Non-secure.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.6 Fault response and synchronization commands

4.6.1 CMD_RESUME(StreamID, SSec, STAG, Action, Abort)



Resumes processing of the stalled transaction identified with the given StreamID and STAG parameter, with the given action parameter, Ac:

Action (Ac)	Result
1 (Retry)	Transaction is retried as though it had just arrived at the SMMU. Configuration and translations are looked up, it might then progress into the system or fault again. The Abort parameter, Ab, is IGNORED.
0: (Terminate)	Transaction is terminated in the manner given by the Abort parameter, Ab. When Ab == 0, the transaction is completed successfully with RAZ/WI semantics. When Ab == 1, an abort/bus error is reported to client. If SMMU_IDR0.TERM_MODEL == 1, the Ab parameter is IGNORED: transaction is terminated with abort. Note: The Abort parameter is analogous to the CD.A configuration for non-stalled terminated transactions. In response to a transaction experiencing a stage 2 fault, the STE configuration provides only two behaviors, whether the transaction is terminated by abort or becomes stalled, but a subsequent CMD_RESUME for the stall will (if supported, as indicated by SMMU_IDR0.TERM_MODEL) allow the stalled transaction to be terminated with RAZ/WI behavior. There is no dependency between STE.S2S and the behavior of the CMD_RESUME, and the Terminate parameter is not limited by STE configuration.

When issued on an SMMU implementation that does not support the Stall model (or where the Stall model has been disabled with NSSTALLD for the Security state of the Command queue), indicated by SMMU_(S_)IDR0.STALL_MODEL == `0b01`, a CERROR_ILL is raised.

The STAG parameter is an opaque token that must be supplied exactly as provided in the corresponding fault event record from which the existence of the stalled transaction is determined.

When issued from the Secure Command queue, SSec controls whether the command is being issued in response to the stall of a Secure or Non-secure StreamID. When issued from the Non-secure Command queue, StreamID is Non-secure, SSec must be zero, because SSec == 1 is ILLEGAL and results in a CERROR_ILL. A CMD_RESUME issued from the Non-secure Command queue does not affect a stalled transaction that originated from a Secure StreamID.

An SMMU must:

- Use the StreamID, SSec and STAG to locate the specific stalled transaction to be resumed.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Differentiate individual transactions in the case of multiple stalled transactions from the same StreamID.
- Verify that a STAG value corresponds to the given StreamID.

If this command is issued with a STAG value that does not correspond to any stalled transaction, or if the transaction does not match the given StreamID, this command has no effect on any transaction (it is effectively a no-op). This can occur in any of these cases:

- There are no stalled transactions from the given StreamID.
- STAG indicates an SMMU resource that is not holding a stalled transaction.
- The STAG indicates an SMMU resource that does hold a stalled transaction, but it is not associated with the given StreamID.

Note: When SMMU_S_IDR1.SECURE_IMPL == 1, the SMMU might be presented with transactions from a Secure stream and a Non-secure stream that have the same StreamID value. The Secure and Non-secure StreamID namespaces are independent so the streams and transactions are unrelated. It is therefore possible for two stalled transactions to exist, one from a Secure stream and the other from a Non-secure stream, which cause an event to be recorded in both Secure and Non-secure Event queues that coincidentally have the same StreamID and STAG value. Despite having the same numeric values, the two stall event records represent independent transactions, as the Security states of the streams are different.

Note: Event records from stalled transactions indicate the StreamID and SubstreamID of the transaction, the SubstreamID is not required to be supplied for this command as STAG locates the specific transaction.

Stalled transactions might be retried with CMD_RESUME in any order, but IMPLEMENTATION DEFINED interconnect ordering rules must still be observed and these might not allow a retried transaction to progress into the system unless a prior stalled transaction is also resumed. For example, two stalled reads from the same StreamID might not be allowed to cross. If the later read is resumed with retry it might still stall until the first read is resumed with retry or terminated, at which point the later read might progress.

ARM expects software to respond to every recorded event record indicating a stall using a CMD_RESUME or a CMD_STALL_TERM, and expects that a CMD_RESUME is only issued in response to a stall event record visible in an Event queue. Software must only issue CMD_RESUME with StreamID and STAG values that have directly been supplied in a stall event record that has not already been subject to a matching CMD_RESUME or CMD_STALL_TERM operation.

The STAG value of a stall event record matched by CMD_RESUME is returned to the set of values that the SMMU might use in future stall event records.

It is CONSTRAINED UNPREDICTABLE whether a matching transaction is affected by this command in the following cases:

- The indicated stalled transaction exists in the SMMU but, at the time of the CMD_RESUME, an event has not yet been made visible to software.
- The encoding of STAG in an implementation allows a CMD_RESUME to target a stalled transaction for which an event was not recorded because it was suppressed as a duplicate, see section 3.12.2.
- The stalled transaction has already been subject to a prior CMD_RESUME or CMD_STALL_TERM.

Note: ARM does not expect software to issue a CMD_RESUME in these circumstances, but an implementation is not required to explicitly prevent an effect.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Consumption of CMD_RESUME(Retry) does not guarantee that the given stalled transaction has already been retried, but does guarantee that, if it has not, it will be retried at a later time. The SMMU will retry the transaction in finite time.

Note: A stalled transaction that has no matching CMD_RESUME response might never be retried.

A retried transaction behaves as though the transaction had just arrived at the SMMU.

Note: The transaction must respect configuration and translation cache invalidations and structure updates that occurred between the time of its original arrival and the retry.

Consumption of CMD_RESUME(Terminate) does not guarantee that the given stalled transaction has already been terminated, but does guarantee that, if it has not, it will be terminated at a later time unless it retries (completing successfully) before being terminated. An implementation ensures that a transaction marked for termination with a CMD_RESUME(Terminate) is terminated in finite time without unbounded delay, if it is not successfully retried before the termination occurs.

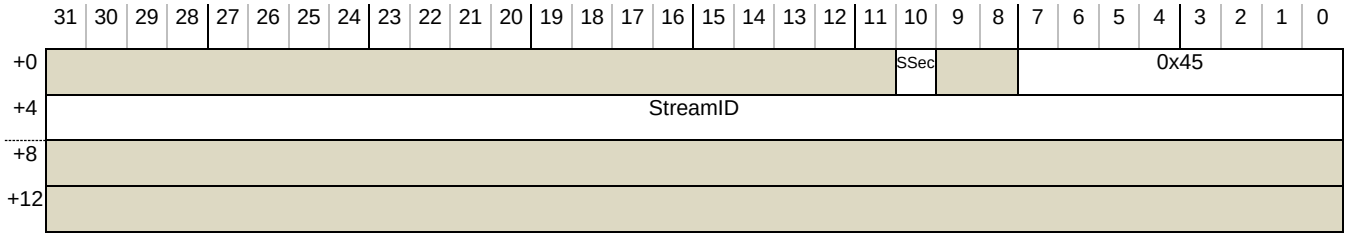
Note: If configuration remains in a state that would cause retries of the transaction to continue to fault, the transaction is guaranteed to be terminated. This also guarantees that a client device might wait for completion of the transaction, and will always eventually make forward progress.

Note: If the transaction is retried before the point of termination, it might complete successfully if its initial fault reason was resolved in the intervening time.

The SMMU does not guarantee response visibility by the client device.

Note: This means that software cannot guarantee that a given transaction has terminated without performing a synchronization involving the originating device originator or interconnect. This would need to be performed in a device- and system-specific manner before changing device configuration or translations in order to ensure that the transaction will not retry successfully with the new configuration. A similar scenario can exist where a system would need to ensure all transactions that are in progress that are buffered arrive at the SMMU (for termination) so that they do not later appear after new configuration or new translation state is applied.

4.6.2 CMD_STALL_TERM(StreamID, SSec)



This command provides a mechanism to mark all stalled transactions originating from StreamID for termination.

Note: This command is equivalent to tracking individual stalls and issuing CMD_RESUME(Terminate) separately.

This command must only be issued after the STE for the StreamID is updated to cause all new incoming transactions to immediately terminate with abort, including any required configuration cache invalidation and synchronization for the STE update. When issued in this condition, any transactions that are outstanding on the stream at the time that this command is observed by the SMMU are guaranteed to terminate with an abort, including the set of transactions that might have become stalled under a previous stream configuration, if the agent controlling the SMMU waits for outstanding transactions to complete before altering the STE of the StreamID to a non-terminating configuration. The mechanism for this wait operation is IMPLEMENTATION DEFINED.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If no stalled transactions from the given stream exist before the command is observed and no transactions become stalled before the command is consumed, this command has no effect.

Consider the following cases:

- The STE is not configured to immediately terminate new transactions. This includes the case when configuration cache maintenance and synchronization for the STE update has not been performed.
- The STE is written in a way that changes such a configuration before all outstanding transactions have completed.

If the CMD_STALL_TERM command is issued when one of these cases apply, it is UNPREDICTABLE whether one of the following occurs:

- Stalled transactions related to the StreamID are terminated.
- Pending stall fault event records are prevented from being recorded.

The STAG values in stall event records associated with stalled transactions affected by CMD_STALL_TERM are returned to the set of values that the SMMU might use in future stall event records.

When issued on an SMMU implementation that does not support the Stall model, or where the Stall model has been disabled with NSSTALLD for the Non-secure Command queue, indicated by SMMU_(S_)IDR0.STALLMODEL == `0b01`, a CERROR_ILL is raised.

A CMD_STALL_TERM issued from the Secure Command queue can terminate transactions originating from either Security state, and SSec indicates the Security state of the StreamID. When issued from the Non-secure Command queue, CMD_STALL_TERM only terminates a stalled transaction if the transaction originated from a StreamID with Non-secure configuration, SSec must be zero. Issuing a CMD_STALL_TERM with SSec == 1 on the Non-secure Command queue is ILLEGAL and results in a CERROR_ILL.

Consumption of a CMD_STALL_TERM does not guarantee that matching stalled transactions have been terminated. When used as described in this section, consumption guarantees that the set of matching stalled transactions will be terminated at a future time. Transactions for the matching StreamID are not affected by this command if they become stalled after the command is consumed.

Note: This behavior matches CMD_RESUME(Terminate).

Note: Early-retry cannot cause stalled transactions to complete successfully before termination occurs, because the STE configuration will terminate such early-retries, and therefore termination is guaranteed.

4.6.2.1 Notes and usage

Some matching stalled transactions might not record events because the events are suppressed, as described in 3.12.2.1. Event queue record visibility semantics are covered in section 3.5.2.

This command is not intended to be used on a stream for which incoming transactions might become stalled during processing of this command because, in this scenario, the point in an ongoing sequence of transactions at which newly-faulting stalling transactions would no longer be matched by an ongoing CMD_STALL_TERM is

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

UNPREDICTABLE. In addition, stall fault events might become visible to software for transactions that were matched and marked for termination by this command.

This command is intended to be used when forcibly de-commissioning or reclaiming a stream controlled by untrusted or unreliable software. It is intended to be issued after the stream has been explicitly configured to immediately terminate future incoming transactions without creating new stalled transactions.

Note: This sequence will shut down a device stream, including removal of stalled transactions:

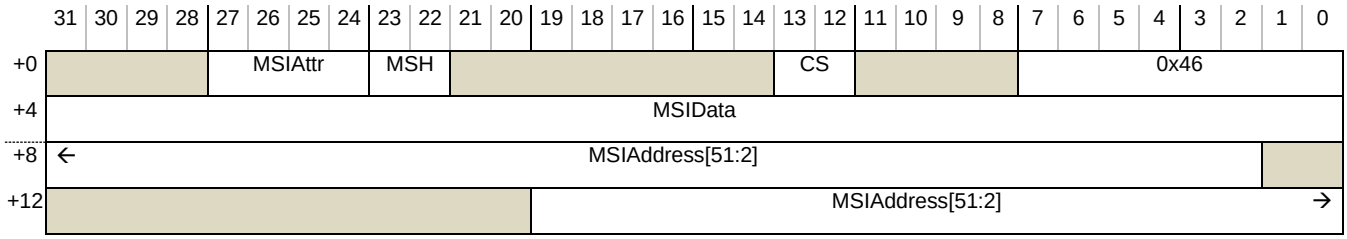
1. Software stops the device from issuing transactions.
2. Transactions are terminated using:
   a. Set STE[i].Config to `0b000` and keep STE[i].V == 1
   b. Issuing a CMD_CFGI_STE(i, j, k) (or other configuration invalidation that covers relevant STE(s)).
   c. Issuing a CMD_SYNC.
   d. At this point, any incoming transactions will terminate and no new transactions will enter stalled state. In addition, the completion of CMD_SYNC ensures visibility of all stall fault event records related to STE[i], see 4.6.3. If a stall event record cannot be written (for example, the queue is full), the transaction is either retried when the queue becomes writable and terminates given the new configuration or is marked for termination, if a retry does not happen before the CMD_STALL_TERM in the next step.

      Note: A stall record (STALL == 1) is not written after this point. Any record that is written relates to new STE configuration, which terminates.
3. CMD_STALL_TERM (i, j)
   a. At this point, remaining stalled transactions are marked for termination, but might not have terminated yet.
4. Software waits for outstanding device transactions to complete.
   a. How this is achieved is IMPLEMENTATION DEFINED.
   b. This step waits for the stalled transactions to terminate and for transactions that are in progress to reach the SMMU and be terminated on arrival.
5. [Optional] CMD_SYNC
   a. If the STE configuration causes incoming transactions to terminate in a non-silent manner (for example, if STE[i].V == 0 instead of STE[i].Config == `0b000`), terminated fault events might have been generated because of the arrival of in-progress transactions during this procedure. This CMD_SYNC ensures visibility (if the Event queue is writable) of the events related to terminated transactions for which an abort has been returned to the client device (see section 4.6.3). If the Event queue is not writable, the completion of the CMD_SYNC means that events for terminated transactions will not become visible after this point.
6. Software discards all event records relating to StreamID ‘i’ in the current set of Event queue records. Thereafter, all events relating to StreamID ‘i’ are from subsequent or newly-created configuration.

This sequence ensures that the Event queue will not subsequently receive any further stall event records associated with the stream and that there are no incomplete transactions still stalled in the SMMU.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

4.6.3 CMD_SYNC(ComplSignal, MSIAddress, MSIData, MSIWriteAttributes)



This command provides a synchronization mechanism for preceding commands. When this command completes, it can raise a completion signal as an optional interrupt and an optional WFE wakeup event. The interrupt might take the form of an MSI (if supported by the implementation). Any operations made observable by the completion of the synchronization operation are observable before the interrupt or event is observable, and before the SMMU_CMDQ_CONS index indicates that the CMD_SYNC has been consumed. Observation of the interrupt or event means that the consumption of the CMD_SYNC is observable.

The ComplSignal parameter, CS, determines the signaling mechanism that notifies host software of the completion of a CMD_SYNC, and can take the following values:

- `0b00`: SIG_NONE: The command takes no further action on its completion. The MSIAddress, MSIData, MSIWriteAttributes parameters are IGNORED.
- `0b01`: SIG_IRQ: The command signals its completion by raising an interrupt. On implementations supporting MSIs, a write containing MSIData is made to the physical address given by MSIAddress, if MSIAddress is non-zero, using memory type attributes from MSIAttr. On implementations that do not support MSIs, MSIAddress and MSIData are IGNORED. On implementations that support wired interrupts, an event on a unique wired interrupt output is asserted on this signal, regardless of the value of MSIAddress.

  Note: This allows the choice of wired or MSIs on implementations that support both. Software can configure an MSI and ignore a wired output, or can disable MSIs and configure an interrupt controller for the wired output.

  Note: See section 3.18. The MSI write might be directed towards an interrupt controller to generate an interrupt, or towards a shared memory location so that the PE can poll, or wait, on the location until the notification appears.

  Note: Where an SMMU can send an MSI and the system allows the MSI write to coherently affect shared cached memory locations, an ARMv8-A PE might use the loss of a reservation on a location as a WFE wakeup event, providing the same semantics as SIG_SEV with SIG_IRQ.
- `0b10`: SIG_SEV: The command sends an Event to the PE similar to SEV.
  - Use of SIG_SEV only sends an event when SMMU_IDR0.SEV is set, otherwise, no event is available to a PE and SIG_SEV is equivalent to SIG_NONE, that is no completion signal is generated.
  - Note: The PE might poll on the SMMU_CMDQ_CONS.RD register in a loop throttled by WFE.
- `0b11`: Reserved: Causes a CERROR_ILL.

The MSI configuration is given by the following parameters:

MSIAttr: Write attribute for MSI, encoded the same as the STE.MemAttr field

MSH: Shareability attribute for MSI write, encoded as:

- `0b00`: Non-shareable
- `0b01`: Reserved, treated as `0b00`.
- `0b10`: Outer Shareable.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- o `0b11`: Inner Shareable.

  Note: This field is IGNORED if MSIAttr specifies a memory type of any-Device or Normal-iNC-oNC, and Shareability is effectively Outer Shareable in these cases.

MSIAddress[51:2]: If this field is non-zero, it configures the physical address that an MSI is sent to when SIG_IRQ is used. Address bits above and below this field are treated as zero. The MSI is a 32-bit word-aligned write of MSIData.

The check for zero applies to the entire specified field span and is not limited to the physical address size (OAS) of an implementation.

If the OAS is smaller than this field, MSIAddress is truncated to the OAS, as described in section 3.4.3.

In SMMUv3.0, MSIAddress[51:48] are RES0.

When a cacheable type is specified in MSIAttr, the allocation and transient hints are IMPLEMENTATION DEFINED.

This command waits for completion of all prior commands and ensures observability of any related transactions through and from the SMMU. Commands in the Command queue that are more recent than a CMD_SYNC do not begin processing until the CMD_SYNC completes.

When following a given command submitted to the SMMU, completion of a CMD_SYNC makes the following guarantees:

Command type	Action
TLB and ATC invalidation commands CMD_TLBI_*, CMD_ATC_INV	TLB invalidates and ATC invalidations are guaranteed to be complete (all matching TLB entries are invalidated) and all transactions in progress that were translated using any of the TLB entries targeted by the invalidates are globally observable to their Shareability domain. The semantics of the completion of TLB invalidation match those of an ARMv8-A PE. ATOS translations that were in progress at the time of the CMD_SYNC and used any of the TLB entries targeted by the invalidations are complete, or re-start from the beginning after the CMD_SYNC completes (see section 3.21.1). Note: No dependency is required on completion of received broadcast TLB invalidation operations. The broadcast invalidation mechanism has its own synchronization and completion mechanisms (for example on AMBA interconnect, a DVM Synch Operation).
Configuration invalidation commands CMD_CFGI_*	Configuration invalidations are guaranteed to be complete (matching cached configuration entries are invalidated) and all transactions that are in progress

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Command type	Action
	that were translated using any of the configuration cache entries targeted by the invalidates are globally observable to their Shareability domain. ATOS translations that were in progress at the time of the CMD_SYNC and used any of the configuration cache entries targeted by the invalidations are complete, or re-start from the beginning after the CMD_SYNC completes (see section 3.21.3).
Prefetch commands CMD_PREFETCH_*	Translation or configuration table walks initiated by any kind of prefetch, including CMD_PREFETCH_*, are affected by TLB or configuration cache invalidates in the same way as any other table walk and might therefore be transitively affected by the completion of a CMD_SYNC of a CMD_TLBI_*/CMD_CFGI_*. See section 3.21.
PRI responses CMD_PRI_RESP	Nothing. The SMMU cannot guarantee response visibility by the endpoint.
Stall resume/termination CMD_RESUME, CMD_STALL_TERM	Nothing. The CMD_RESUME and CMD_STALL_TERM commands complete by the time they are consumed.
Synchronization commands CMD_SYNC	The guarantees of the prior CMD_SYNC have been met. CMD_SYNCs complete in order. The MSI writes of any prior CMD_SYNC are visible to their Shareability domain, where the commands are consumed from the same queue. CMD_SYNC is not required to affect MSI writes originating from sources other than prior CMD_SYNC completion signals, or completion signals for CMD_SYNC commands related to a different Command queue.

In addition, completion of a CMD_SYNC ensures the following behavior:

- An unrecorded fault event record relating to a client transaction terminated by the SMMU (either immediately on a fault or after stalling) whose abort or termination response could have been observed by the client device before the start of the CMD_SYNC, is guaranteed to have either:
    - Become visible in the relevant Event queue (given the visibility semantics of section 3.5.2).
    - Been discarded, if it could not commit to write to the queue, because the queue is (full or disabled. In this case the unrecorded fault event record will never become visible.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

    - Note: This rule ensures that if software performs a CMD_SYNC after notification from a device that all of its outstanding transactions are complete, it is guaranteed that no termination fault records will later become visible from the device stream after the CMD_SYNC completes.
    - This rule applies however the termination of the transaction was performed, whether it was immediate because the Terminate fault model was used or whether the transaction originally stalled and was later terminated because a retry encountered a new configuration, or an update of SMMU_(S_)CR0.SMMUEN transitioned the field to 0. In the case of a retry encountering new configuration, any recorded event relates to the new configuration not the original stall.
- Where a particular interconnect does not return a response to a terminated client transaction, all committed unrecorded fault records corresponding to transactions internally terminated before the start of the CMD_SYNC are made visible. Uncommitted fault records either commit to the Event queue and are made visible, or if they cannot be committed, are discarded and will not later become visible.
- Event records written to the Event queue after the completion of a CMD_SYNC are guaranteed to be generated from configurations or translations visible to the SMMU after any invalidation completed by the CMD_SYNC has taken effect. No events relating to out-of-date configuration will later become visible.
- When an unrecorded fault event record exists for a stalled transaction affected by TLBI or CFGI invalidations completed by a CMD_SYNC, the completion also affects the event record in one of the following ways:
    - The record commits to write and is made visible in the relevant Event queue.
    - If it cannot commit to write to the queue, because the queue is full or disabled, the transaction is retried when the queue is next writable, if it has not terminated or otherwise completed before that time, and this would lead to the generation of a new event record instead of the original one, if the configuration has changed. If it retries successfully, no event is recorded for the transaction, because the original event is stale. This adheres to the previous rule. The original event record pertaining to prior invalidated configuration must not be recorded after the CMD_SYNC completes.
- HTTU caused by completed client transactions and completed ATOS translations is made visible, to the extent required to its Shareability domain, by completion of a CMD_SYNC, see section 3.13.4.

There is no requirement for:

- A CMD_SYNC submitted to the Non-secure Command queue to affect Secure traffic or event visibility. However, a CMD_SYNC submitted to the Secure Command queue affects Non-secure traffic or event visibility when the CMD_SYNC completes prior commands in the Secure Command queue that operate on Non-secure structures.
- A CMD_SYNC to affect any recorded stalled transaction, or to cause an unrecorded stalled transaction to be retried, except where necessary to record an event relating to newly-changed configuration.

Note: A TLB or structure invalidation command has no explicit ordering against prior or subsequent non-CMD_SYNC commands. If software requires one set of invalidations to be guaranteed complete before beginning a second set of invalidations, a CMD_SYNC is required to separate the two sets.

If the completion of a prior CMD_ATC_INV cannot be guaranteed by a CMD_SYNC because of a PCIe protocol error, such as a timeout, the CMD_SYNC might cause a CERROR_ATC_INV_SYNC command error to be raised, see sections 7.1 and 3.9.1.2. If a CMD_SYNC raises a CERROR_ATC_INV_SYNC error:

- The CMD_SYNC is not complete and has not been consumed, that is, SMMU_(S_)CMDQ_CONS.RD remains pointing at the CMD_SYNC that raised the error. No completion signals are sent.
- The completion guarantees of the CMD_SYNC have not been met.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- An outstanding CMD_ATC_INV command is one that was submitted to the command queue before the CMD_SYNC and that has not been completed by a previous successful CMD_SYNC, and:
    - An UNKNOWN set of outstanding CMD_ATC_INV commands has timed out and will never complete.
    - An UNKNOWN set of outstanding CMD_ATC_INV commands might be in the process of completing successfully, but are not guaranteed to have completed successfully.
- All other operations that the CMD_SYNC would otherwise have completed are not guaranteed to have completed, but will be completed by a new CMD_SYNC that is successfully consumed after being re-submitted to the queue after the error is resolved and command processing is resumed.

Note: For example, a CMD_TLBI_* command, that was processed prior to the CMD_SYNC that caused the error, might still be being processed and is guaranteed to still take effect; a CERROR_ATC_INV_SYNC does not cause such commands to be terminated and there is no requirement for them to be re-submitted after resolving the error.

4.7 Command Consumption summary

Command type	Consumption means:
TLB and ATC invalidation commands CMD_TLBI_*, CMD_ATC_INV	Nothing
Configuration invalidation commands CMD_CFGI_*	Nothing
Prefetch commands CMD_PREFETCH_*	Nothing
PRI responses CMD_PRI_RESP	Nothing
Stall resume/termination CMD_RESUME, CMD_STALL_TERM	CMD_RESUME and CMD_STALL_TERM have individual completion guarantees that have been met.
Synchronization commands CMD_SYNC	The completion guarantees of CMD_SYNC have been met.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

5 Data structure formats

All SMMU configuration data structures, that is Stream table entries and Context descriptors, are little-endian. Translation table data might be configured for access in either endianness on some implementations, see CD.ENDI, STE.S2ENDI and SMMU_IDR0.TTENDIAN.

All non-specified fields are RES0, Reserved, and software must set them to zero. An implementation either:

- Detects non-zero values in non-specified fields and considers the structure invalid.
- Ignores the reserved field entirely.

All specified fields are required to be checked against the defined structure validity rules in each of the STE and CD sections.

A configuration or programming error or invalid use of any of the fields in these structures might cause the whole structure to be deemed invalid, where specified. Any transaction from a device that causes the use of an invalid structure will report an abort back to the device and will log an error event, the nature of which is specific to the nature of the misconfiguration.

A structure is used when it is indicated by an STE (or in the case of an STE itself, when an incoming transaction selects it from the Stream table, by StreamID).

The memory attribute set by SMMU_(S_)CR1.TABLE_{SH,OC,IC} is used when fetching the following structures:

- Level 1 Stream table descriptor.
- Stream table entry.

The attribute set by STE.{S1CIR,S1COR,S1CSH} is used when fetching a Level 1 Context descriptor and Context descriptor.

See section 3.21.3 for information on structure access, caching and invalidation rules.

See section 16.2 for implementation notes on caching and interactions between structures.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

5.1 Level 1 Stream Table Descriptor

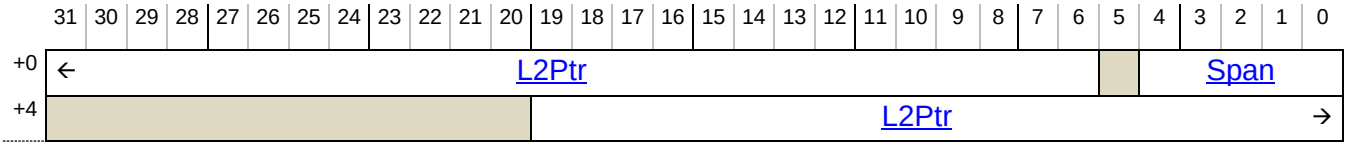


Figure 13: Level 1 Stream table descriptor

Bits	Name	Meaning
[4:0]	Span	2^n size of Level 2 array and validity of L2Ptr: Span: Meaning 0: L2Ptr is invalid. The StreamIDs matching this descriptor are all invalid. 1-11: Level 2 array contains $2^{(Span-1)}$ STEs [(1)] 12-31: Reserved, behaves as 0
[51:6]	L2Ptr[51:6]	Pointer to start of Level-2 array. Address bits above and below the field range are treated as 0. Bits L2Ptr[N:0] are treated as 0 by the SMMU, where `N == 5 + (Span - 1)` Note: The Level 2 array is therefore aligned to its size by the SMMU. In SMMUv3.0, bits [51:48] are RES0. See section 3.4.3 for behavior of addresses beyond the OAS or physical address range.

(1) Span must be within the range of 0 to (SMMU_STRTAB_BASE_CFG.SPLIT + 1), that is it must stay within the bounds of the Stream table split point.

Incoming StreamIDs that select a descriptor with Span == 0, or a Span set to a reserved value, or a Span set to an out of bounds value given the split point, or those that select a valid Level 1 descriptor but are outside of the level 2 range described by Span, are deemed invalid. A transaction causing a Stream table lookup that does not reach a valid STE is terminated with an abort to the client device and optionally records an event, see SMMU_(S_)CR2.RECINVSID and C_BAD_STREAMID.

When an L1STD is changed, the non-leaf form of CMD_CFGI_STE is the minimum scope of invalidation command required to invalidate SMMU caches of the L1STD entry. Depending on the change, other STE invalidations might be required, for example:

- Changing an inactive L1STD with Span == 0 to a non-zero active Span (introducing a new section of level-2 Stream table) requires an invalidation of the L1STD only. As no STEs were reachable for StreamIDs within the span, none require invalidation.
- Changing an active L1STD with Span != 0 to an inactive L1STD (decommissioning a span of Stream table) requires an invalidation of the L1STD as well as invalidation of cached STEs from the affected span. Either multiple non-leaf CMD_CFGI_STE commands, or a wider scope such as CMD_CFGI_STE_RANGE or CMD_CFGI_ALL is required.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

5.2 Stream Table Entry

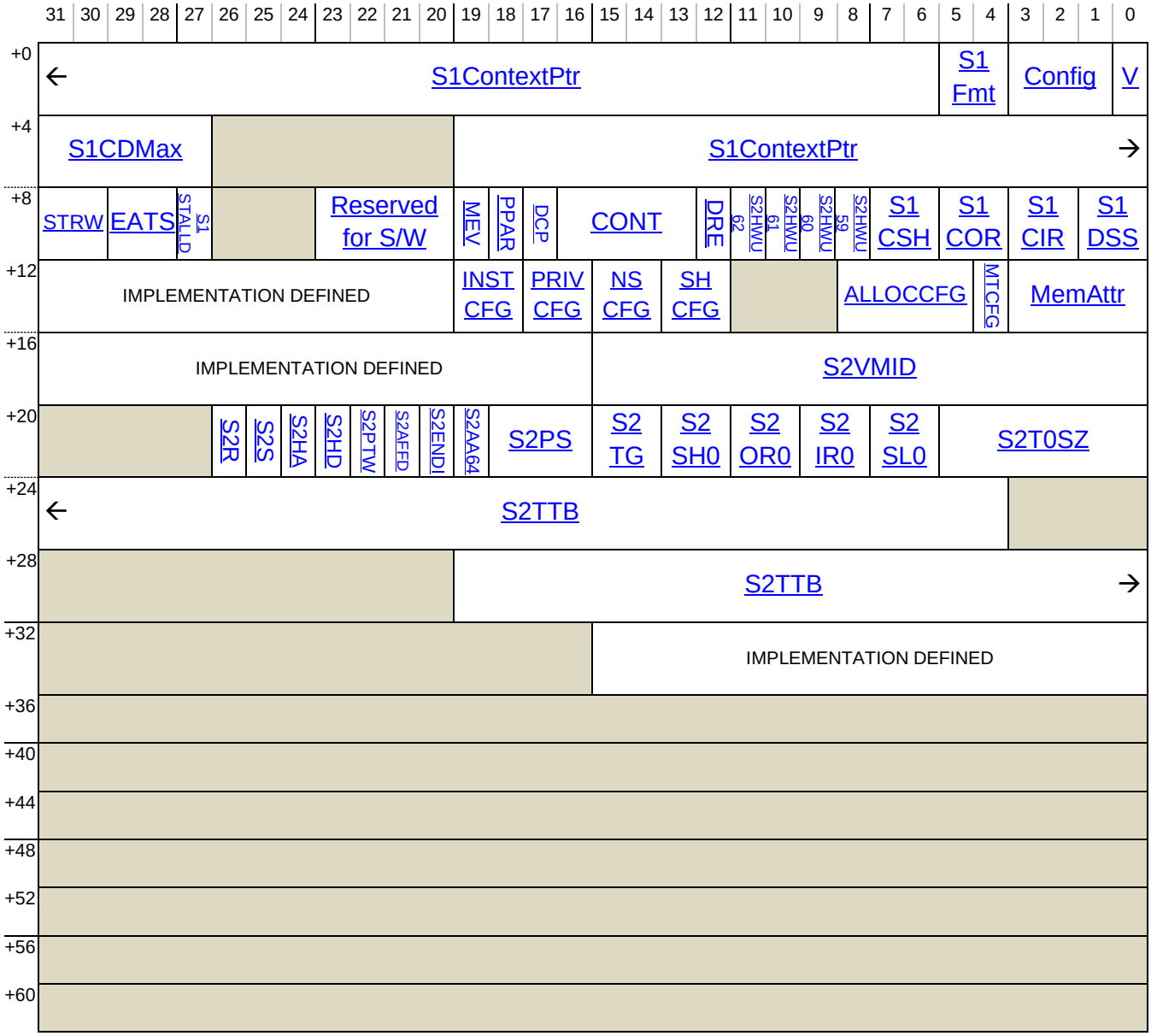


Figure 14: Stream Table Entry fields

STE fields follow the convention of an S1 prefix for fields related to stage 1 translation, an S2 prefix for fields relating to stage 2 translation, and neither for fields unrelated to a specific translation stage.

In a valid STE, that is where STE.V == 1,

- All fields with an S2 prefix (with the exception of S2VMID) are IGNORED when stage 2 bypasses translation (Config[1] == 0).
- All fields with an S1 prefix are IGNORED when stage 1 bypasses translation (Config[0] == 0).

The validity conditions in field descriptions are written with the assumption that the field is not IGNORED because of a disabled stage of translation.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: Additionally, the validity check of STE.EATS might assert that STE.S2S == 0 even if stage 2 is in bypass. That is STE.S2S is not IGNORED.

All undefined fields, and some defined fields where explicitly noted, are permitted to take RAZ/WI behavior in an embedded implementation providing internal storage for Stream table entries, see section 3.16. Permitted differences for such an embedded implementation that does not store STEs in regular memory are marked *EI* in this section.

Note: This allows such implementations to avoid storing bits that do not affect the SMMU behavior.

Invalid or contradictory configurations are marked ILLEGAL in field descriptions. Use of an ILLEGAL STE behaves as described for STE.V == 0.

<table>
<tr><th>Bits</th><th>Name</th><th>Meaning</th></tr>
<tr><td>[0]</td><td>V</td><td>Valid:
`0b0`: Structure contents are invalid. Other STE fields are IGNORED.
`0b1`: Structure contents are valid. Other STE fields behave as described.
Care must be taken when updating an STE when V == 1 as updates might race against SMMU fetching the structure, see section 3.21.

Device transactions that select an STE with V == 0 are terminated with an abort reported back to the device and a C_BAD_STE event is recorded.

ATS Translation Requests that select an STE with V == 0 are immediately completed with CA status. No event is recorded.

When SMMU_CR0.ATSCHK == 1, ATS Translated transactions are checked against STE configuration. Those selecting an invalid STE (with ILLEGAL configuration, or V == 0) are terminated with an abort reported back to the device. No event is recorded. See 3.9.1 for more information on ATS-related transactions.</td></tr>
<tr><td>[3:1]</td><td>Config[2:0]</td><td>Stream configuration:
<table>
<tr><th>Value</th><th>Traffic can pass?</th><th>Stage 1</th><th>Stage 2</th><th>Notes</th></tr>
<tr><td>`0b000`</td><td>No</td><td>-</td><td>-</td><td>Report abort to device, no event recorded.</td></tr>
<tr><td>`0b0xx`</td><td>No</td><td>-</td><td>-</td><td>Reserved (behaves as `0b000`)</td></tr>
<tr><td>`0b100`</td><td>Yes</td><td>Bypass</td><td>Bypass</td><td>EATS value effectively `0b00`</td></tr>
<tr><td>`0b101`</td><td>Yes</td><td>Translate</td><td>Bypass</td><td>S1* valid</td></tr>
</table></td></tr>
</table>

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning

`0b110`	Yes	Bypass	Translate	S2* valid
`0b111`	Yes	Translate	Translate	S1* and S2* valid

If stage 1 is not implemented (SMMU_IDR0.S1P == 0), it is ILLEGAL to set Config[2:0] == `0b1x1`.

If stage 2 is not implemented (SMMU_IDR0.S2P == 0), it is ILLEGAL to set Config[2:0] == `0b11x`.

If stage 2 is implemented but the STE is reached from the Secure Stream table, it is ILLEGAL to set Config[2:0] == `0b11x` because stage 2 is only supported for NS-EL1 configurations.

Note: When stage 1 is configured to translate, see the descriptions for S1DSS and S1Fmt for substream configuration.

In an EI, Config[0] is permitted to be RAZ/WI if stage 1 is not implemented and Config[1] is permitted to be RAZ/WI if stage 2 is not implemented.

When no translation stages are enabled (`0b100`), ATS Translation Requests (and Translated traffic, if SMMU_CR0.ATSCHK == 1) are denied as though EATS == `0b00`; the actual value of the EATS field is IGNORED. Such a Translation Request causes F_BAD_ATS_TREQ and Translated traffic causes F_TRANSL_FORBIDDEN.

When Config[2:0] == `0b000`, an ATS Translation Request is denied with UR status and no event is recorded. When Config[2:0] == `0b000` and SMMU_CR0.ATSCHK == 1, ATS Translated transactions are terminated with an abort and no event is recorded.

[5:4] S1Fmt

S1ContextPtr points to:

S1CDMax	S1Fmt	
0	xx	One Context descriptor (S1Fmt is IGNORED).
Non-zero	`0b00`	Two or more Context descriptors in a linear table indexed by SubstreamID[S1CDMax - 1:0]. The table is aligned to its size.S1ContextPtr[S1CDMax + 5:6] are RES0 and these bits of the address are taken to be zero by the SMMU.
	`0b01`	2-level table with 4KB L2 leaf tables. L1 table contains 1-16384 L1CD pointers (128KB maximum) to 4KB tables of 64 CDs. If S1CDMax ≤6, only index #0 of the L1 table is used.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning		
				Otherwise, the L1 table is indexed by SubstreamID[S1CDMax - 1:6]. The L2 table is indexed by SubstreamID[5:0]. L2 tables are 4KB-aligned: L1CD.L2Ptr[11:0] are taken to be zero by the SMMU. L1 tables are aligned to their size, or 64 bytes, whichever is greater. If S1CDMax > 9, S1ContextPtr[S1CDMax - 4:6] are RES0 and these bits are taken to be zero by the SMMU.
			`0b10`	2-level table with 64KB L2 leaf tables. L1 table contains 1-1024 L1CD pointers (8KB maximum) to 64KB tables of 1024 CDs. If S1CDMax≤10, only index #0 of the L1 table is used. Otherwise, the L1 table is indexed by SubstreamID[S1CDMax - 1:10]. The L2 table is indexed by SubstreamID[9:0]. L2 tables are 64KB-aligned: L1CD.L2Ptr[15:12] are RES0 and L1CD.L2Ptr[11:0] are taken to be zero by the SMMU. L1 tables are aligned to their size, or 64 bytes, whichever is greater. If S1CDMax > 13, S1ContextPtr[S1CDMax - 8:6] are RES0 and these bits are taken to be zero by the SMMU.
			`0b11`	Reserved (behaves as `0b00`)

When multiple substreams are supported and enabled, the supported range is 2-1024K substreams in all table layouts.

If Config[0] == 0 (stage 1 disabled), this field is IGNORED and the supply of a SubstreamID with a transaction causes the transaction to be terminated with an abort, recording C_BAD_SUBSTREAMID. This is the case for full stream bypass (Config[2:0] == `0b100`), or stage 2-only translation (Config[2:0] == `0b110`).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		If S1CDMax == 0 (substreams disabled) or SMMU_IDR1.SSIDSIZE == 0 (substreams unsupported), this field is IGNORED and S1ContextPtr points to one CD. If substreams are supported and enabled and stage 1 is enabled and SMMU_IDR0.CD2L == 0, it is ILLEGAL to set S1Fmt != `0b00` (as two-stage tables are not supported). In an EI, this field is permitted to be RAZ/WI if stage 1 is not implemented or substreams are unsupported or SMMU_IDR0.CD2L == 0. Note: If stage 2 is configured, S1ContextPtr is an IPA. If S1Fmt indicates a 2-level table, the pointers in the first-level table are also IPAs. Otherwise, the pointers are PAs. Note: S1Fmt == `0b00` can be used for a single CD, or to support multiple CDs for substreams. A 4KB page will hold 64 CDs, and a 64KB page will hold 1024 CDs. Note: ARM expects that the effective number of CDs in simultaneous use is practically limited to the number of distinct address spaces which is limited by the number of ASIDs. With S1Fmt == `0b01`, 64K CDs can be accommodated with an 8KB L1 table and multiple 4KB L2 tables.
[51:6]	S1ContextPtr [51:6]	Stage 1 Context descriptor pointer bits [51:6]. S1ContextPtr bits above and below this range are implied as zero in CD address calculations. If Config[0] == 0 (stage 1 disabled), this field is IGNORED. In an EI, this field is permitted to be RAZ/WI if stage 1 is not implemented. If Config[1] == 1 (stage 2 enabled), this pointer is an IPA translated by stage 2 and the programmed value must be within the range of the IAS. Otherwise, this pointer is a PA, is not translated, and must be within the range of the OAS. See section 3.4.3 for allowed behavior of an out-of-range value in S1ContextPtr. In SMMUv3.0, STE[51:48] are RES0.
[63:59]	S1CDMax	Number of CDs pointed to by S1ContextPtr, $2^{S1CDMax}$. If SMMU_IDR1.SSIDSIZE == 0, this field is IGNORED. A transaction is not permitted to supply a SubstreamID. If a transaction does so, it will be terminated and a C_BAD_SUBSTREAMID event recorded. Note: ARM expects that an implementation that does not support substreams will also not have a mechanism to supply a SubstreamID, so this situation cannot not occur. In an EI, this field is permitted to be RAZ/WI if stage 1 is not implemented or if SMMU_IDR1.SSIDSIZE == 0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		If Config[0] == 0 (stage 1 disabled), this field is IGNORED, Otherwise, when this field is 0, the substreams of the STE are disabled and one CD is available. (The minimum useful number of substreams is 2.) Any transaction with a SubstreamID will be terminated with an abort and a C_BAD_SUBSTREAMID event recorded. When this field is > 0, a specific span of substreams are in use. A transaction with a SubstreamID will be terminated with an abort and a C_BAD_SUBSTREAMID event recorded if the SubstreamID is outside of the range specified by this field. The allowable range is 0 to SMMU_IDR1.SSIDSIZE inclusive. Other values are ILLEGAL. The behavior of a transaction without a SubstreamID when S1CDMax > 0 is governed by the S1DSS field.
[65:64]	S1DSS	Default Substream: When substreams are enabled (S1CDMax != 0), this field determines the behavior of a transaction or translation request that arrives without an associated substream:

`0b00`	Terminate. An abort is reported to the device and the F_STREAM_DISABLED event is recorded.
`0b01`	Bypass stage 1 as though Config[0] == 0 The transaction can cause a stage 1 Address Size fault if the input address size exceeds the IAS, see section 3.4 for details. If the configuration enables stage 2 translation, the address is then translated as an IPA if a stage 1 Address Size fault did not occur. Note: This behavior is identical to a transaction through an STE with Config[0] == 0. Note: Such a transaction does not fetch a CD, and therefore does not report F_CD_FETCH, C_BAD_CD or a stage 2 Translation-related fault with CLASS == CD.
`0b10`	Transactions that do not include a substream are translated using the CD associated with Substream 0, which becomes unavailable for use by transactions that include a substream. Transactions that include a substream and select Substream 0 are terminated. An abort is reported to the device and the F_STREAM_DISABLED event is recorded. System software must associate traffic with non-zero substreams

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		<table><tr><td></td><td>because Substream 0 is not available for use.</td></tr><tr><td>0b11</td><td>Reserved (behaves as 0b00)</td></tr></table> If Config[0] == 0 (stage 1 disabled) or S1CDMax == 0 (substreams disabled) or SMMU_IDR1.SSIDSIZE == 0 (substreams unsupported), this field is IGNORED. In an EI, this field is permitted to be RAZ/WI if stage 1 is not implemented or if SMMU_IDR1.SSIDSIZE == 0. Note: PCIe traffic might include a PASID TLP prefix, or might be issued without it. Consequently, it is possible for some transactions to supply a substream while others from the same endpoint do not. Note: This field affects ATS Translation Requests, which can be caused to skip stages of translation as described in section 13.6.3 and 13.6.4. See section 3.9.2 for information on changing configuration that affects ATS translations that could be cached in an Endpoint. For ATS Translation Requests, if the cases described in 0b00 and 0b10 lead to termination, the Translation Request is terminated with a CA and no event is recorded.
[67:66]	S1CIR	S1ContextPtr memory Inner Region attribute: <table><tr><td>0b00</td><td>Normal, non-cacheable</td></tr><tr><td>0b01</td><td>Normal, Write-Back cacheable, Read-Allocate</td></tr><tr><td>0b10</td><td>Normal, Write-Through cacheable</td></tr><tr><td>0b11</td><td>Normal, Write-Back cacheable, no Read-Allocate</td></tr></table> Note: Read allocation is a hint in the same way as in the ARMv8-A architecture, and it is IMPLEMENTATION DEFINED whether it has any effect. Note: Because CDs are read-only there is no configuration for cache allocation on write. The value of the write-allocation hint provided for a CD read is IMPLEMENTATION DEFINED. S1CIR, S1COR and S1CSH set the memory access attributes that access the Context descriptor (and Level-1 CD table pointers, if relevant) through S1ContextPtr. If Config[1] == 1 (stage 2 enabled), the CD is loaded from IPA space and the attributes are combined with those from the stage 2 translation descriptor of the page mapping the accessed IPA, otherwise, these attributes are used directly. In an EI, S1CIR, S1COR and S1CSH are permitted to be RAZ/WI if stage 1 is not implemented.
[69:68]	S1COR	S1ContextPtr memory Outer Region attribute:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		<table><tr><td>0b00</td><td>Normal, Non-cacheable</td></tr><tr><td>0b01</td><td>Normal, Write-Back cacheable, Read-Allocate</td></tr><tr><td>0b10</td><td>Normal, Write-Through cacheable</td></tr><tr><td>0b11</td><td>Normal, Write-Back cacheable, no Read-Allocate</td></tr></table>
[71:70]	S1CSH	S1ContextPtr memory Shareability attribute: <table><tr><td>0b00</td><td>Non-shareable</td></tr><tr><td>0b01</td><td>Reserved (behaves as 0b00)</td></tr><tr><td>0b10</td><td>Outer Shareable</td></tr><tr><td>0b11</td><td>Inner Shareable</td></tr></table> Note: If both S1CIR and S1COR == 0b00, selecting normal Non-cacheable access, the Shareability of access to CDs is taken to be Outer Shareable regardless of the value of this field.
[72]	S2HWU59	When SMMU_IDR3.PBHA == 1, this bit controls the interpretation of bit [59] of the stage 2 translation table final-level (page or block) descriptor: • 0: Bit [59] is not interpreted by hardware for an IMPLEMENTATION DEFINED purpose. • 1: Bit [59] has IMPLEMENTATION DEFINED hardware use. This bit is IGNORED when PBHA are not supported (SMMU_IDR3.PBHA == 0). This bit is RES0 in SMMUv3.0. Note: Stage 2 translations do not have the Hierarchical Attribute Disable (HAD) control present for stage 1 and the stage 2 HWU bits therefore do not have a relation to HAD in the same way as for stage 1.
[73]	S2HWU60	Similar to S2HWU59, but affecting descriptor bit [60].
[74]	S2HWU61	Similar to S2HWU59, but affecting descriptor bit [61].
[75]	S2HWU62	Similar to S2HWU59, but affecting descriptor bit [62].
[76]	DRE	Destructive Read Enable. Some implementations might support transactions with data-destructive effects which intentionally cause cache lines to be invalidated, without writeback even if they are dirty, such as destructive reads or cache invalidation operations, see section 3.22. Note: The invalidation side-effect is not required for correctness of this class of transaction, but if performed might cause stale data to be made visible. • 0b0: A device request to consume data using a read and invalidate transaction is transformed to a read without a data-destructive side-effect. An Invalidate Cache Maintenance Operation is transformed to a CleanInvalidate operation. • 0b1: A device request to consume data using a read and invalidate

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		transaction is permitted to destructively read the requested location. An Invalidate Cache Maintenance Operation is permitted without transformation. Both of these behaviors are dependent on the correct page permissions as described in sections 3.22.2 and 16.7.2.2. This bit is IGNORED on implementations that do not support this class of transactions. Otherwise, this bit only applies when at least one stage of translation is applied (if STE.S1DSS configuration or Config == `0b100` causes all stages to be bypassed, a read and invalidate or Invalidate Cache Maintenance Operation is permitted regardless of the value of this bit). This bit is RES0 in SMMUv3.0.
[80:77]	CONT	Contiguous Hint This field provides a hint to SMMU caching structures that a fetched STE is identical to its neighbours within a particular span of StreamIDs, and that a cache of the STE might be matched for any future lookup of StreamID for translation purposes within this span. The field is a 4-bit unsigned value specifying the span size. This field does not affect configuration invalidation. Software must ensure every STE in the required range is targeted by appropriate CMD_CFGI_* commands, regardless of the value of the field. When CONT == 0, the STE is an individual STE bordered by STEs not considered identical. Otherwise, the span is defined as a contiguous block of 2^{CONT} STEs starting at a StreamID for which StreamID[CONT-1:0]=0. All defined fields, except for CONT, in all STEs within the stated span must be identical, otherwise the result of an STE lookup has one of the following CONSTRAINED UNPREDICTABLE behaviors: • The requested STE is used as it is represented in the Stream table. • A neighbouring STE within the CONT span is used. • An F_CFG_CONFLICT event is reported and the transaction or translation request that led to the STE lookup is aborted. Note: During the process of marking STEs as members of a contiguous span (or taking away such hints), the STEs might differ by their CONT field values. Note: ARM expects that the CONT value is changed in valid STEs and that the STEs in the span are not made invalid before changing CONT. This ensures that the STEs in the span remain identical. A span greater than the size of the Stream table is capped by the SMMU by the size of the Stream table, and does not allow StreamIDs beyond the Stream table span to match an STE. For a given valid STE in a 2-level Stream table, software must not program

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		STE.CONT to a span greater than the Span field of the L1STD that locates the STE. Otherwise, use of a StreamID within the range of the erroneous STE.CONT field is CONSTRAINED UNPREDICTABLE and gives rise to either one of these behaviors: • The transaction is terminated and a C_BAD_STREAMID error is raised, if the StreamID is outside of the range represented by the Span of the corresponding L1STD. Note: This is the expected behavior when using a StreamID outside of an L1STD.Span range. • The transaction uses any STE from the same Security state as the stream. In an EI, this field is permitted to be RAZ/WI.

[81]	DCP	Directed Cache Prefetch.
		
		Some implementations might support directed cache prefetch hint operations which are a standalone hint transaction, or a read/write transaction with hint side-effect, that changes the cache allocation in a part of the cache hierarchy that is not on the direct path to memory. This class of operation does not include those with data-destructive side-effects, see section 3.22.
		
		• `0b0`: Directed cache prefetch operations are inhibited. A transaction input with hint side-effect is stripped of the hint. A standalone hint operation completes successfully having no effect on the system.
		• `0b1`: A directed cache prefetch operation is permitted as follows:
		  o A transaction with hint side-effect is performed if the final translation permissions permit the transaction.
		  ▪ Note: This permits a write with side-effect to progress if permissions grant write access, otherwise a Permission fault occurs.
		  o A standalone hint operation is performed if the final permissions grant either read or write or execute permission for the requested address, otherwise the hint completes successfully having no effect on the system.
		
		This bit is IGNORED on implementations that do not support this class of transactions. Otherwise, this bit only applies when at least one stage of translation is applied (if STE.S1DSS configuration or Config == `0b100` causes all stages to be bypassed, a directed cache prefetch is permitted regardless of the value of this bit).
		
		This bit is RES0 in SMMUv3.0.

[82]	PPAR	PRI Page request Auto Responses 0: Auto-generated responses on PRI queue overflow do not include a PASID TLP prefix. 1: Auto-generated responses on PRI queue overflow include a PASID TLP prefix if

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		permitted. See sections 8.1 and 8.2 for the conditions that permit a PASID TLP prefix to be used on an auto-generated response. If SMMU_IDR0.PRI == 0 or SMMU_IDR1.SSIDSIZE == 0, this field is RES0. If SMMU_IDR3.PPS == 1, this field is IGNORED and the behavior of an auto-generated response is the same as described for PPAR == 1.
[83]	MEV	Merge Events arising from terminated transactions from this stream: 0: Do not merge similar fault records 1: Permit similar fault records to be merged The SMMU might be able to reduce the usage of the Event queue by coalescing fault records that share the same page granule of address, access type and SubstreamID. Setting MEV == 1 does not guarantee that faults will be coalesced. Setting MEV == 0 causes a physical SMMU to prevent coalescing of fault records, however, a hypervisor might not honour this setting if it deems a guest to be too verbose. Note: Software must expect, and be able to deal with, coalesced fault records even when MEV == 0. In an EI, this field is permitted to be RAZ/WI if the implementation does not merge events. See section 7.3.1 for details on event merging.
[87:84]	Res S/W	Reserved for software use, this field is IGNORED by the SMMU. In an EI, storage must be provided for this field.
[91]	S1STALLD	Stage 1 Stall Disable: 0: Allow stalling fault model for stage 1 (configured in CD) 1: Disallow stalls to be configured for Stage 1 (faults terminate immediately) If stage 1 is not implemented (SMMU_IDR0.S1P == 0), this bit is RES0. If stage 1 is not enabled (Config[0] == 0), this bit is IGNORED. Otherwise, if stage 1 is enabled and SMMU_(S_)IDR0.STALL_MODEL is not `0b00`, it is ILLEGAL to set this field to 1 because the Stall model is not supported or not configurable. When the Stall model is configurable, this bit must be set for StreamIDs associated with stall-unsafe system topologies or for PCIe clients.
[93:92]	EATS	Enable PCIe ATS translation and traffic. This field enables responses to ATS Translation Requests, and when SMMU_CR0.ATSCHK == 1, controls whether ATS Translated traffic can pass into the system. `0b00` — ATS Translation Requests are returned unsuccessful or aborted (UR) and F_BAD_ATS_TREQ is recorded. Additionally, when SMMU_CR0.ATSCHK == 1, Translated traffic

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning	
			associated with the StreamID of the STE is prevented from bypassing the SMMU. Such traffic is terminated with an abort and a F_TRANSL_FORBIDDEN event is recorded.
		`0b01`	Full ATS: ATS Translation Requests are serviced by translating at all enabled stages of translation. Translated traffic from the StreamID of the STE is allowed to bypass (regardless of SMMU_CR0.ATSCHK). In SMMUv3.1 implementations, this configuration is ILLEGAL if EATS is not IGNORED and Config[1] == 1 and S2S == 1. In SMMUv3.0 implementations, this configuration is ILLEGAL if EATS is not IGNORED and S2S == 1. It is CONSTRAINED UNPREDICTABLE whether or not this check of S2S occurs when Config[1] == 0. ARM recommends that S2S == 1 causes EATS == `0b01` to be ILLEGAL only when Config[1] == 1.
		`0b10`	Split-stage ATS: ATS Translation responses return the IPA output of stage 1 translation to the Endpoint or ATC. Subsequent Translated transactions are generated by the Endpoint with IPAs and these undergo stage 2 translation in the SMMU, see section 13.6.3 for details. This configuration must only be used when: • The stream has both stage 1 and stage 2 translation (Config[2:0] == `0b111`) • S2S == 0 • SMMU_IDR0.NS1ATS == 0 • SMMU_CR0.ATSCHK == 1 If any of the following hold, the STE is ILLEGAL (if it is not IGNORED): • Config[2:0] != `0b111` • S2S == 1. • SMMU_IDR0.NS1ATS == 1. If Config[2:0] == `0b111` && S2S == 0 && NS1ATS == 0, but

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		SMMU_CR0.ATSCHK == 0 then EATS == `0b10` configuration behaves as `0b00`. Note: See section 13.6.3 and 13.6.4 which describes interaction of this field with STE.S1DSS.
		`0b11` Reserved (behaves as `0b00`)

This field is RES0 if the STE is Secure, and the effective value of EATS is `0b00`.

This field is IGNORED if any of the following hold:

- SMMU_IDR0.ATS == 0
  - ATS is not supported by the SMMU implementation. No ATS requests can be made, nor Translated traffic passed.
- Config[1:0] == `0b00`
  - When Config[2:0] == `0b100`, the effective value of EATS is `0b00`.The responses and events recorded for Translation Requests and Translated transactions are as described in this table for EATS == `0b00`.
  - When Config[2:0] == `0b000`, Translation Requests are silently terminated with UR and, if SMMU_CR0.ATSCHK == 1, Translated transactions are silently aborted.

Incoming PCIe traffic marked Translated is only checked against the effective value of EATS if SMMU_CR0.ATSCHK == 1, otherwise traffic marked Translated bypasses the SMMU.

Incoming ATS Translation Requests are always checked against the effective value of EATS.

The behavior of EATS == `0b10` configuration is dependent on SMMU_CR0.ATSCHK; see section 3.9.2 for details on changing ATS configuration. An implementation is permitted to cache ATSCHK in configuration caches, so if ATSCHK is changed while STEs exist with EATS == `0b10` it is UNPREDICTABLE as to whether the behavior on receipt of new requests related to those STEs is as though EATS == `0b00` or EATS == `0b10`.

In an EI, this field is permitted to be RAZ/WI if SMMU_IDR0.ATS == 0.

Note: There is no dependency between EATS being non-zero and the ability for a CD to select the stall fault model with CD.S == 1. ARM expects that the Stall model is not enabled for PCIe-related streams both at stage 2 (STE.S2S == 0) and at stage 1 (CD.S == 0) and that STE.STALLD == 1 is used if necessary to ensure that a CD

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		cannot use CD.S == 1 when the CD configuration is not managed the highest-privilege entity in the system. This expectation is present regardless of whether the stream uses ATS or not. If STE.STALLD == 0 and CD.S == 1, the Stall model might (if supported) cause PCIe traffic to stall upon fault.
[95:94]	STRW	StreamWorld control: Selects the translation regime and associated Exception level controlling this stream, in conjunction with the Security state of the STE as defined by being fetched using the Non-secure Stream table or the Secure Stream table, when stage 1 translation is used. This field is RES0 if stage 1 is not implemented (SMMU_IDR0.S1P == 0) and the effective StreamWorld is determined by STE.Config[1]. This field is RES0 if SMMU_IDR0.Hyp == 0 and the STE is in the Non-secure Stream table. If STE.Config[1] == 1 (stage 2 translation enabled), STRW is IGNORED (if it is not already RES0) and the effective StreamWorld is NS-EL1. If STE.Config[1] == 0 (stage 2 bypass) and STE.Config[0] == 0 (stage 1 bypass), STRW is IGNORED (if it is not already RES0) as no translations are performed. Note: The STRW field is not the same as the StreamWorld, which is an attribute whose effective value might be influenced by STRW in some configurations, but not in others. See section 3.3.3 for a definition of StreamWorld. When Config[0] == 1 and Config[1] == 0 (stage 1-only translation), the StreamWorld is determined as follows. The StreamWorld affects three things: • The tagging of resulting TLB entries, that is the translation regime, and ASID or VMID • The number of translation tables used in the CD for stage 1 • The Permissions model used in stage 1 translation table descriptors.

Which Stream table?	STRW	SMMU_CR2.E2H	Resulting TLB entries tagged with		TTBs used *(c)*	TTD perms *(d)*	Permitted Stage 1 TT formats *(e)*
			Stream World	ASID, VMID			
NS *(a)*	`0b00`	X	**NS-EL1**	ASID+ VMID	TTB0, TTB1	All	AArch32, AArch64
	`0b10`	0	**EL2**	None (f)	TTB0	Privileged only	AArch32, AArch64

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning

		1	**EL2-E2H**	ASID	TTB0, TTB1	All	AArch64
	`0bx1`	X	Reserved, ILLEGAL				
S *(b)*	`0b00`	X	**Secure**	ASID	TTB0, TTB1	All	AArch32, AArch64
	`0b01`	X	**EL3**	None (f)	TTB0	Privileged only	AArch64
	`0b1x`	X	Reserved, ILLEGAL				

Configuration inputs are the left-hand columns in white. The result of the configuration is listed in the right-hand shaded columns.

Notes:

(a) The STE is reached using the Non-secure Stream table. When SMMU_IDR0.Hyp == 1, STRW selects NS-EL1, EL2 or EL2-E2H, Reserved values render the STE ILLEGAL. When SMMU_IDR0.Hyp == 0, STRW is RES0 (because EL2 is not supported and the resulting TLB entries of the STE are tagged as NS-EL1).

(b) The STE is reached using the Secure Stream table and Config[0]=1 (stage 1 enabled). STRW selects 'Secure' or 'EL3':

- Secure tags TLB entries as Secure with an ASID and must be selected for Secure streams used by:
  - Secure software on an ARMv7-A host PE, or ARMv8-A host PE whose EL3 runs in AArch32
  - Secure-EL1 software on an ARMv8-A host PE whose EL3 runs in AArch64
- EL3 tags TLB entries as EL3 without an ASID and must be selected for Secure streams used by EL3 software on an ARMv8-A host PE whose EL3 runs in AArch64. There are no ASIDs in an AArch64 EL3 translation regime.

(c) EL2 and EL3 regimes use only one translation table. TTB1 is unused in these configurations.

(d) The format of access permissions in a stage 1 translation table descriptor is affected when located from a configuration with StreamWorld == EL2 or StreamWorld == EL3, consistent with the ARMv8 Translation System [4]. Specifically, the AP[1] bit (of the AP[2:1] field) is ignored and treated as if it were 1 because privilege checks are ignored for EL2 and EL3 (AArch64) translations. However, EL2-E2H translations maintain privileged/non-privileged checks in the same manner as EL1.

(e) Only some stage 1 translation table formats are valid in each StreamWorld, consistent with the PE. Valid combinations are depicted in the table and in the CD.AA64 description. Selecting an inconsistent combination of StreamWorld and CD.AA64 (for example, using AArch32 page tables to represent an AArch64 EL3 translation regime) causes the CD to be ILLEGAL.

(f) In a regime that lacks ASIDs to differentiate address spaces, all CDs are

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		considered equivalent (similar to two CDs with the same ASID value) even if referenced using different STEs. This implies that SubstreamIDs cannot differentiate address spaces in EL2/EL3 StreamWorlds. See 5.4.1 for restrictions on permitted differences between CDs in such StreamWorlds. The StreamWorld tag also determines how TLB entries respond to incoming broadcast TLB invalidations and TLB invalidation SMMU commands, see section 3.17 for details.
[99:96]	MemAttr	If MTCFG == 1, MemAttr provides memory type override for incoming transactions. The encoding matches the VMSAv8-64 stage 2 MemAttr[3:0] field, except that the following encodings are Reserved (not UNPREDICTABLE) and behave as Device-nGnRnE: • `0b0100` • `0b1000` • `0b1100` If MTCFG == 0 or SMMU_IDR1.ATTR_TYPES_OVR == 0, this field is RES0.
[100]	MTCFG	Memory Type configuration: 0: Use incoming type or Cacheability 1: Replace incoming type or Cacheability with that defined by MemAttr field When SMMU_IDR1.ATTR_TYPES_OVR == 0, this field is RES0 and the incoming Memory Type is used. It is IMPLEMENTATION DEFINED whether MTCFG applies to streams associated with PCIe devices or whether the incoming Memory Type is used for such streams regardless of the field value.
[104:101]	ALLOCCFG	Allocation hints override: `0b0xxx` — Use incoming RA, WA, TR hints `0b1RWT` — Hints are overridden to given values: Read Allocate == R Write Allocate == W Transient == T When overridden by this field, for each of RA/WA and TR, both inner and outer hints are set to the same value. Because it is not architecturally possible to express hints for types that are any-Device or Normal-Non-cacheable, this field has no effect on memory types that are not Normal-WB or Normal-WT, whether such types are provided with transaction from the client device or overridden using MTCFG/MemAttr. Note: A value of `0b1001` encodes Transient with no-allocate. When the SMMU ensures output attribute consistency (see section 13.1.6), no-allocate is considered to be non-transient. As there is no value of stage 1/stage 2 attribute with which

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		Transient-no-allocate could combine to cause an allocating Transient attribute to be output, `0b1001` is functionally equivalent to `0b1000` (non-Transient, no-allocate). When SMMU_IDR1.ATTR_TYPES_OVR == 0, this field is RES0 and the incoming Allocation hints are used. It is IMPLEMENTATION DEFINED whether ALLOCCFG applies to streams associated with PCIe devices or whether the incoming allocation hints are used for such streams regardless of the field value.
[109:108]	SHCFG	Shareability override: `0b00` Non-shareable `0b01` Use incoming Shareability attribute `0b10` Outer shareable `0b11` Inner shareable Architecturally, any-Device and Normal-iNC-oNC are OSH and Shareability is only variable where cacheable types are used. This field has no effect on memory types that do not contain Normal-{i,o}WB or Normal-{i,o}WT, whether such types are provided with transaction from the client device or overridden using MTCFG/MemAttr. When SMMU_IDR1.ATTR_TYPES_OVR == 0, this field is RES0 and the incoming Shareability attribute is used. It is IMPLEMENTATION DEFINED whether SHCFG applies to streams associated with PCIe devices or whether the incoming Shareability attribute is used for such streams regardless of the field value.
[111:110]	NSCFG	Bypass NS attribute configuration: `0b00` Use incoming NS attribute `0b01` Reserved, behaves as `0b00`. `0b10` Secure `0b11` Non-secure NSCFG is IGNORED when the STE is in the Non-secure Stream table. NSCFG is IGNORED when the STE is in the Secure Stream table and stage 1 translation is enabled. In this case the final NS attribute is determined wholly by the translation process (see CD.NSCFGx, the TTD.NSTable and TTD.NS bits); the input NS attribute and STE.NSCFG are not used. Otherwise, when the STE is in the Secure Stream table and bypass is selected

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		(Config[2:0] == `0b100`): • When SMMU_IDR1.ATTR_PERMS_OVR == 0, this field is RES0 and the incoming NS attribute is used. • When SMMU_IDR1.ATTR_PERMS_OVR == 1, this field determines the NS attribute of bypassing transactions. Note: The function of this field is not related to the CD.NSCFG{0,1} fields (whose purpose is to affect the NS property of the page table walk).
[113:112]	PRIVCFG	User/privileged attribute configuration: `0b00` Use incoming PRIV attribute `0b01` Reserved (behaves as `0b00`) `0b10` Unprivileged `0b11` Privileged When SMMU_IDR1.ATTR_PERMS_OVR == 0, this field is RES0 and the incoming PRIV attribute is used.
[115:114]	INSTCFG	Inst/Data attribute configuration: `0b00` Use incoming INST attribute `0b01` Reserved (behaves as `0b00`) `0b10` Data `0b11` Instruction When SMMU_IDR1.ATTR_PERMS_OVR == 0, this field is RES0 and the incoming INST attribute is used. INSTCFG only affects translations for reads, the SMMU considers incoming writes to be Data regardless of this field. See section 13.1.2.
[127:116]		IMPLEMENTATION DEFINED per-stream configuration. (For example, QoS overrides for configuration access, data streams.)
[143:128]	S2VMID	Virtual Machine Identifier: Marks TLB entries inserted because of translations located through this STE, differentiating them from translations belonging to different virtual machines. When stage 2 is implemented (SMMU_IDR0.S2P == 1), translations resulting from a StreamWorld == NS-EL1 configuration are VMID-tagged with S2VMID, just as in a PE TLB, when stage 1 (Config[1:0] == `0b01`), stage 2 (Config[1:0] == `0b10`) or both stages (Config[1:0] == `0b11`) provide translation. When an implementation supports only 8-bit VMIDs, that is whenSMMU_IDR0.VMID16 == 0, it is ILLEGAL for bits S2VMID[15:8] to be non-zero unless this field is IGNORED. When stage 2 is not implemented (SMMU_IDR0.S2P == 0) or when Config[1:0] == `0b00` or StreamWorld != NS-EL1, no VMID tagging of TLB entries occurs and

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		S2VMID[15:0] is IGNORED. In an EI, this field is permitted to be RAZ/WI if stage 2 is not implemented. In an EI implementing stage 2 with 8-bit VMIDs, S2VMID[15:8] are permitted to be RAZ/WI. If 16-bit VMIDs are supported by an implementation, the full VMID[15:0] value is used regardless of AA64. ARM expects that legacy and AArch32 hypervisor software using 8-bit VMIDs will write zero-extended 8-bit values in the VMID field in this case. See section 3.17 for more information on ASID and VMID TLB tagging. Changes to the S2VMID field of an STE are not automatically reflected in cached translations, which must be subjected to separate TLB maintenance. Note: This field might be supplied by an implementation in transactions to the memory system for IMPLEMENTATION DEFINED purposes.
[159:144]		IMPLEMENTATION DEFINED.
[165:160]	S2T0SZ	Size of IPA input region covered by stage 2 translation table. This field is 4 bits ([3:0]) with bits [5:4] IGNORED when stage 2 translation is AArch32 (S2AA64 == 0). The input region size is calculated the same as in ARMv8-A VMSA/AArch32 VTCR. When S2AA64 == 1, this field is 6 bits and the region size is calculated the same as for the ARMv8 VTCR_EL2. If stage 2 translation is enabled (Config[1] == 1), legal values of S2SL0 and S2TG relate to this field as defined by the ARMv8 translation system [4]. Note: This field is IGNORED when stage 2 is implemented but not enabled (Config[1] == 0). When STE.S2AA64 == 1: • The maximum valid value is 39. • In SMMUv3.0, the minimum valid value for this field is ‘64-IAS’. • In SMMUv3.1: o If STE.S2TG selects a 4KB or 16KB granule, the minimum valid value for this field is `MAX(16, 64-IAS)`. o If STE.S2TG selects a 64KB granule, the minimum valid value for this field is ‘64-IAS’. Note: If AArch32 is implemented, `IAS == MAX(40, OAS)`, otherwise IAS == OAS, see section 3.4. When STE.S2AA64 == 0, this field encodes a value from -8 to 7 as defined by the ARMv8-A translation system [4]. Note: When S2AA64 == 0 all 4-bit values are valid, therefore it is not possible to use a value outside the valid range but it is still possible for the S2T0SZ to be inconsistent

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		with S2SL0. In SMMUv3.0 implementations, the use of a value out of range of these maximum and minimum valid values is CONSTRAINED UNPREDICTABLE and has one of the following behaviors: • The STE becomes ILLEGAL. • The STE is not ILLEGAL and the effective value used by the translation is the maximum permitted value (if the programmed value is greater than the maximum permitted value) or the minimum permitted value (if the programmed value is less than the minimum permitted value). In SMMUv3.1 implementations, an STE is treated as ILLEGAL if it contains a S2T0SZ value out of range of these maximum and minimum values. The usable range of values is further constrained by a function of the starting level set by S2SL0 and, if S2AA64 == 1, granule size set by S2TG as described by the ARMv8 translation system. Use of a value of S2T0SZ that is inconsistent with the permitted range (given S2SL0 and S2TG) is ILLEGAL. Note: The SMMU behavior differs from the PE translation system, in which the legal range of VTCR_EL2.T0SZ values can also depend on whether EL1/stage 1 uses AArch32 or AArch64. This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0.
[167:166]	S2SL0	Starting level of stage 2 translation table walk. This field is encoded the same as the ARMv8 VTCR_EL2.SL0 field and the encoding depends on S2TG. If stage 2 translation is enabled (Config[1] == 1), it is ILLEGAL for this field to be inconsistent with S2T0SZ and S2TG. This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0.
[169:168]	S2IR0	Inner region Cacheability for stage 2 translation table access: `0b00` Non-cacheable `0b01` Write-back Cacheable, Read-Allocate, Write-Allocate `0b10` Write-through cacheable, Read-Allocate `0b11` Write-back cacheable, Read-Allocate, no Write-Allocate Because the only time a translation table is written by the SMMU is when HTTU is in use, and because the atomic update might require data to be already cached, it is IMPLEMENTATION DEFINED as to whether `0b01` and `0b11` differ.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		HTTU might require TTDs to be cached. It is IMPLEMENTATION DEFINED as to whether HTTU is performed when `0b00` or `0b10` are in use. ARM recommends that software uses `0b01` or `0b11` when HTTU is enabled for this translation table unless the behavior of an implementation is otherwise known. This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0.
[171:170]	S2OR0	Outer region Cacheability for stage 2 translation table access, same as for S2IR0
[173:172]	S2SH0	Shareability for stage 2 translation table access: `0b00` Non-shareable `0b01` Reserved, behaves as 00. `0b10` Outer Shareable `0b11` Inner Shareable Note: If both S2IR0 and S2OR0 == `0b00`, selecting normal Non-cacheable access, the Shareability of translation table access is taken to be OSH regardless of the value of this field. This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0.
[175:174]	S2TG	Stage 2 Translation Granule size: `0b00` 4KB `0b01` 64KB `0b10` 16KB `0b11` Reserved This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0. Otherwise, if stage 2 translation is disabled (Config[1] == 0), this field is IGNORED. Otherwise, if stage 2 translation is enabled (Config[1] == 1), • If S2AA64 == 0, this field is RES0 and the effective S2TG value is 4KB. • If S2AA64 == 1, this field must only select a granule supported by the SMMU, as described in SMMU_IDR5, and use of an unsupported size or Reserved value is ILLEGAL. • It is ILLEGAL for S2T0SZ and S2SL0 to be inconsistent with the effective value of this field, as described in [4].
[178:176]	S2PS	Physical address Size `0b000` 32 bits

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning

0b001	36 bits
0b010	40 bits
0b011	42 bits
0b100	44 bits
0b101	48 bits
0b110	52 bits In SMMUv3.0 implementations, this value is Reserved and behaves as 0b101.
0b111	In SMMUv3.0 implementations, this value is Reserved and behaves as 0b101. In SMMUv3.1 implementations, this value is Reserved and behaves as 0b110.

Software must not rely on the behavior of Reserved values.

This field determines the Physical address size for the purpose of Address Size fault checking the output of stage 2 translation, see section 3.4.

This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0.

When S2AA64 == 0, S2PS is IGNORED and the effective stage 2 output address size is taken as 40 bits.

When S2AA64 == 1, the effective stage 2 output address size is given by:

```
Effective_S2PS = MIN(STE.S2PS, SMMU_IDR5.OAS);
```

The effective S2PS size is capped to the OAS, which is a maximum of 52 bits in SMMUv3.1 and 48 bits in SMMUv3.0.

When S2AA64 == 1, setting this field to any value greater than the cap defined here behaves as though this field equals the cap size. Software must not rely on this behavior.

Note: This includes use of a Reserved value.

An Address Size fault occurs if stage 2 translation outputs an address having bits [51:Effective_S2PS] non-zero.

An address greater than 48 bits in size can only be output from stage 2 when the effective S2PS is 52 and a 64KB granule is in use for that translation table. In SMMUv3.0 addresses are limited to 48 bits.

Note: If a granule smaller than 64KB is used, the address output bits [51:48] are treated as 0b0000, so in this case no Address Size fault can occur if the effective

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		STE.S2PS is greater than or equal to 48 bits.
[179]	S2AA64	0: Stage 2 translation table is AArch32 (LPAE) 1: Stage 2 translation table is AArch64 This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0. If stage 2 translation is enabled (Config[1] == 1), it is ILLEGAL to select: • AArch32 tables when AArch32 tables are not supported (SMMU_IDR0.TTF[0] == 0). • AArch64 tables when AArch64 tables are not supported (SMMU_IDR0.TTF[1] == 0). Note: The stage 2 translation table permissions are interpreted slightly differently between AArch64 and AArch32 (LPAE) format tables, for example, an AArch64 stage 2 table can encode an execute-only page permission whereas an AArch32 stage 2 table cannot, see [4] for more information.
[180]	S2ENDI	Stage 2 translation table endianness: 0: Little Endian 1: Big Endian If Stage 2 translation is enabled (Config[1] == 1), it is ILLEGAL for this field to select an unimplemented endianness (as indicated by SMMU_IDR0.TTENDIAN). This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0.
[181]	S2AFFD	Stage 2 Access Flag Fault Disable: When HTTU is not in use at stage 2 because S2HA == 0 or HTTU is not supported, this flag determines the behavior on access of a stage 2 page whose descriptor has AF == 0: 0: An Access flag fault occurs (behavior controlled by S2R. S2S bits) 1: An Access flag fault never occurs; the TTD.AF bit is considered to be always 1. When S2HA == 1, this flag is IGNORED. This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0.
[182]	S2PTW	Protected Table Walk: For an STE configured for translation at both stages, a stage 1 translation table walk access or CD fetch access made to a stage 2 page with any Device type is terminated and recorded as a stage 2 Permission fault if this bit is set. Note: This might provide early indication of a programming error. 0: CD fetch and Stage 1 translation table walks allowed to any valid stage 2 address 1: CD fetch or Stage 1 translation table walks to stage 2 addresses mapped as any Device are terminated. A stage 2 Permission fault is recorded.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

<table>
<tr><th>Bits</th><th>Name</th><th>Meaning</th></tr>
<tr><td></td><td></td><td>If Config[1:0] != 0b11, this field is IGNORED.
This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0.</td></tr>
<tr><td>[184:183]</td><td>S2HA, S2HD</td><td>Hardware Translation Table Update of stage 2 Access/Dirty flags:
<table>
<tr><th>S2HA</th><th>S2HD</th><th></th></tr>
<tr><td>0</td><td>0</td><td>HTTU disabled</td></tr>
<tr><td>1</td><td>0</td><td>Update of Access flag enabled</td></tr>
<tr><td>0</td><td>1</td><td>Reserved – behaves as S2HA == S2HD == 0</td></tr>
<tr><td>1</td><td>1</td><td>Update of Access flag and Dirty state of the page enabled</td></tr>
</table>
If stage 2 translation is enabled (Config[1] == 1),
• It is ILLEGAL to set S2HA or S2HD if S2AA64 == 0.
• It is ILLEGAL to set S2HA if SMMU_IDR0.HTTU == 0b00.
• It is ILLEGAL to set S2HD if SMMU_IDR0.HTTU == 0b00 or 0b01.
These fields are RES0 when stage 2 is not implemented.
In an EI, S2HA is permitted to be RAZ/WI if stage 2 is not implemented or SMMU_IDR0.HTTU == 0b00. In an EI, S2HD is permitted to be RAZ/WI when stage 2 is not implemented or SMMU_IDR0.HTTU == 0b0x.

Note: If HTTU is enabled when S2OR0 or S2IR0 indicate Non-cacheable memory, behavior is IMPLEMENTATION DEFINED, see 3.15 above. A system might only be able to perform atomic updates using cacheable normal memory, or might implement other means for doing so.</td></tr>
<tr><td>[186:185]</td><td>{S2R,S2S}</td><td>Stage 2 fault behavior: Record and Stall.

See section 5.5 below for a description of fault configuration.

When Config[1] == 0 (Stage 2 disabled), {S2R, S2S} are IGNORED.
This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0.</td></tr>
<tr><td>[243:196]</td><td>S2TTB[51:4]</td><td>Address of Translation Table base. Address bits above and below the field range are treated as zero.

Bits [(x-1):0] are treated as if all the bits are zero, where x is defined by the required alignment of the translation table as given by ARMv8-A [4].
Note: The SMMU effectively aligns the value in this field before use.
In addition, a 64-byte minimum alignment on starting-level translation table addresses is imposed when S2TG selects 64KB granules and the effective S2PS value indicates 52-bit output. In this case bits [5:0] are treated as zero.
If Stage 2 translation is enabled (Config[1]=1), it is ILLEGAL for the address in S2TTB to be outside the range described by the effective S2PS value. It is ILLEGAL for the</td></tr>
</table>

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		address in S2TTB to be outside of a 48-bit range when S2TG selects a granule smaller than 64KB. Note: Where the effective S2PS < 52, an address greater than the effective S2PS that is still within the defined range of this field (up to bit [51]) causes an ILLEGAL STE. Additionally, a 52-bit address can only be used in S2TTB when a 64KB granule is configured for stage 2. This field is RES0 when stage 2 is not implemented, that is when SMMU_IDR0.S2P == 0. In an EI, the high-order bits of TTB outside of the PA size (SMMU_IDR5.OAS) are permitted to be RAZ/WI. If these bits are not implemented as RAZ/WI, they must store the full address field and correctly support the ILLEGAL address range-check described above. In SMMUv3.0, STE[243:240] are RES0.
[271:256]		IMPLEMENTATION DEFINED

If V == 1 and Config == `0b100` (Stream Bypass), the S1* and S2* fields are IGNORED. The following fields are used to apply attributes to the Bypass transactions:

- MTCFG/MemAttr, ALLOCCFG, SHCFG.
- NSCFG (Ignored unless STE is selected from Secure Stream table).
- PRIVCFG, INSTCFG.
    - Note: it is system-dependent as to whether INST/DATA attributes have any effect on the path to the memory system.

If V == 1 and Config == `0b000` (disabled), the CONT field is permitted to be obeyed. The remaining fields contain no relevant configuration and are IGNORED. Transactions selecting such an STE will be silently terminated with an abort.

For more details on the memory attribute and permissions overrides (MTCFG, ALLOCCFG, SHCFG, NSCFG, PRIVCFG, INSTCFG), see section 13.

An STE or L1STD that is successfully fetched might be cached by the SMMU in any state, therefore any modification, commissioning or decommissioning of an STE must be followed by a CMD_CFGI_STE command. A failed fetch (F_STE_FETCH) does not cause an STE or L1STD to be cached.

When cached, an STE is uniquely identified by StreamID, and SEC_SID when two Security states are supported.

Note: An STE from one Stream table index X is unrelated to the STE at index X of the Stream table of the other Security state.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If an implementation supports bypass transactions (because STE.Config == `0b100`) by creating 'identity-mapped' TLB entries, the presence of these entries is not visible to software. Changing STE.Config does not require explicit TLB invalidation.

Note: STE configuration invalidation is required for any alteration to an STE.

StreamWorld (as determined from the STRW field, SMMU_CR2.E2H, Config[1:0] and the Security state of the Stream table that fetches the STE) controls the tagging of TLB entries, so a change of the StreamWorld of a stream simply makes lookups performed for the stream fail to match TLB entries of a prior StreamWorld or translation regime. Software must consider the possibility of such TLB entries still being present if a prior StreamWorld configuration is returned to, unless explicit invalidation has occurred. ARM recommends that TLB entries that are made unreachable by a change in StreamWorld are invalidated after the change to avoid their unanticipated use by a future configuration that happens to match the old StreamWorld, ASID and VMID (if appropriate).

Note: The following STE properties affect the interpretation of a CD located through the STE:

- StreamWorld.
- S1STALLD.
- AArch32 stage 2 translation (STE.Config[1] == 1 and STE.S2AA64 == 0)

When stage 2 translation is enabled (Config[1] == 1), the correct setting of the stage 2 table walk starting level, S2SL0, is dependent on the granule size, S2TG, and the required IPA address size (S2T0SZ). All three fields must be set in a manner consistent with the equivalent ARMv8-A fields. A mismatch causes the STE to be considered ILLEGAL. In ARMv8-A, an inconsistency between SL0, TG and T0SZ is reported as a Translation fault. In addition, a S2TTB base address outside the range indicated by the effective STE.S2PS value makes the STE ILLEGAL. In ARMv8-A, an inconsistency between TTBR and PS is reported as an Address Size fault.

The following STE fields are permitted to be cached as part of a translation or TLB entry and, when altered, software must perform explicit invalidation of any TLB entry that might have cached these fields, after performing STE structure cache invalidation:

- S2TTB.
- S2PTW.
- S2VMID.
- S2T0SZ.
- S2IR0.
- S2OR0.
- S2SH0.
- S2SL0.
- S2TG.
- S2PS.
- S2AFFD.
- S2HA.
- S2HD.
- S2ENDI.
- S2AA64.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Alteration of other STE fields does not require invalidation of TLB entries. IMPLEMENTATION DEFINED STE fields might have IMPLEMENTATION DEFINED invalidation requirements.

Note: Invalidation of an STE also implicitly invalidates cached CDs fetched through the STE.

Stage 2 configuration from STEs with the same S2VMID is considered interchangeable by the SMMU. Software must ensure that all STEs containing the same S2VMID value are identical for non- IGNORED fields permitted to be cacheable as part of a TLB entry, in the list in this section. Fields that are IGNORED because of a stage 2 bypass (Config[1] == 0) are not covered by this rule.

Note: As the properties of TLB or translation cache entries inserted from an STE depend on the fields listed here, a difference would cause TLB entries to be cached with different properties. It would be UNPREDICTABLE as to whether a TLB entry was cached using a particular STE sharing an S2VMID value, therefore the properties returned by a general TLB lookup under the given VMID become UNPREDICTABLE.

5.2.1 Validity of STE

The following expression indicates whether an STE is considered valid or ILLEGAL, for the purposes of determining a configuration error (C_BAD_STE). Further checks are required (for example, on the extent of an incoming SubstreamID with respect to STE.S1CDMax) after an STE is discovered to be valid.

```
PA_RANGE    = (STE.S2AA64 == 0) ? 40 : MIN(S2PS, SMMU_IDR5.OAS);
                // PA range is 40 bits for AA32 tables, else effective PS size
EFF_S2TG    = (STE.S2AA64 == 0) ? 4KB : STE.S2TG;
// Note:  In these cases, the STRW field is ignored or reserved so the
//        value itself cannot contribute to the STE being ILLEGAL.  However,
//        writing a non-zero value to a RES0 field might still make the STE
//        ILLEGAL.  The logic to check RES0/undefined fields is not depicted here.
STRW_UNUSED = (SMMU_IDR0.S1P == 0) || (!STE_SECURE && SMMU_IDR0.Hyp == 0) ||
              (STE.Config[1] == 1) || (STE.Config[1:0] == 0b00);

// See section 4.12.  Assuming SMMU_S_IDR1.SECURE_IMPL == 1, NSSTALLD may affect
// the Non-secure STALL_MODEL:
EFFECTIVE_SMMU_IDR0_STALL_MODEL = SMMU_S_IDR0.STALL_MODEL == 0b00 ?
                                    (SMMU_S_CR0.NSSTALLD == 0 ? 0b00 : 0b01)
                                    : SMMU_S_IDR0.STALL_MODEL;

STE_ILLEGAL = !STE.V || ((Config[2:0] != 0b0XX) && (
                (SMMU_IDR0.S1P == 0 && STE.Config[0] == 1) ||
                (SMMU_IDR0.S2P == 0 && STE.Config[1] == 1) ||
                (SMMU_IDR0.S2P == 1 && STE.Config[1] == 1 && STE_SECURE) ||
                (SMMU_IDR1.SSIDSIZE != 0 && STE.S1CDMax != 0 &&
                      STE.Config[0] == 1 && SMMU_IDR0.CD2L == 0 &&
                            (STE.S1Fmt == 0b01 || STE.S1Fmt == 0b10)) ||
                (!STE_SECURE && SMMU_IDR0.ATS == 1 && Config[1:0] != 0b00 && (
                      (STE.EATS == 0b10 &&
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

```
                (STE.Config[2:0] != 0b111 || STE.S2S == 1 ||
                      SMMU_IDR0.NS1ATS == 1)) ||
            (STE.EATS == 0b01 && STE.S2S == 1 &&
                (STE.Config[1] == 1 || CONSTR_UNPRED_EATS_S2S)))) ||
        (!STE_SECURE && !STRW_UNUSED &&
            (STE.STRW == 0b01 || STE.STRW == 0b11)) ||
        (STE_SECURE && !STRW_UNUSED && STE.STRW > 0b01) ||
        (STE.Config[0] == 1 && (
            (STE.S1STALLD == 1 && !STE_SECURE &&
                EFFECTIVE_SMMU_IDR0_STALL_MODEL != 0b00) ||
            (STE.S1STALLD == 1 && STE_SECURE &&
                SMMU_S_IDR0.STALL_MODEL != 0b00) ||
            (SMMU_IDR1.SSIDSIZE != 0 &&
                STE.S1CDMax > SMMU_IDR1.SSIDSIZE) ||
            /* See 3.4.3: */
            CONSTR_UNPRED_CHECK_PTR_SIZE_FAIL(STE.S1ContextPtr))) ||
        (STE.Config[1] == 1 && (
            (EFFECTIVE_SMMU_IDR0_STALL_MODEL == 0b01 && STE.S2S == 1) ||
            (EFFECTIVE_SMMU_IDR0_STALL_MODEL == 0b10 && STE.S2S == 0) ||
            (STE.S2AA64 == 1 &&
                !GRANULE_SUPPORTED(STE.S2TG, SMMU_IDR5)) ||
            (STE.S2AA64 == 0 && SMMU_IDR0.TTF[0] == 0) ||
            (STE.S2AA64 == 1 && SMMU_IDR0.TTF[1] == 0) ||
            ((STE.S2HA || STE.S2HD) && (STE.S2AA64 == 0)) ||
            ((STE.S2HA || STE.S2HD) && (SMMU_IDR0.HTTU == 0b00)) ||
            (STE.S2HD && SMMU_IDR0.HTTU == 0b01) ||
            ADDR_OUTSIDE_RANGE(STE.S2TTB, PA_RANGE) ||
            (S2T0SZ_OUTSIDE_VALID_RANGE(STE.S2T0SZ, STE.S2AA64) &&
                CONSTR_UNPRED_S2T0SZ_OOR_ILLEGAL == YES) ||
            !WALK_CONFIG_CONSISTENT(STE.S2SL0, STE.S2T0SZ, STE.S2TG,
                                    STE.S2AA64) ||
            (SMMU_IDR0.TTENDIAN == 0b10 && STE.S2ENDI == 1) ||
            (SMMU_IDR0.TTENDIAN == 0b11 && STE.S2ENDI == 0))) ||
        (SMMU_IDR0.S2P == 1 && STE.Config[1:0] != 0b00 &&
            STE.Config[2] == 1 && !STE_SECURE &&
            // For a Non-secure STE with translation enabled,
            // STRW_UNUSED implies an effective StreamWorld of NS-EL1.
            // This expression is then “is NS-EL1”:
            (STRW_UNUSED || STE.STRW == 0b00) &&
            SMMU_IDR0.VMID16 == 0 &&
            STE.S2VMID[15:8] != 0x00)
));
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

5.3 Level 1 Context Descriptor

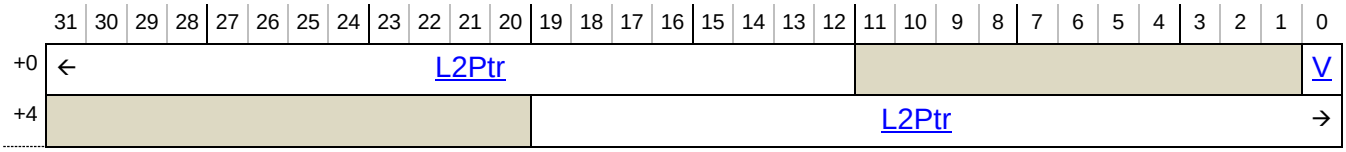


Figure 15: Level 1 Context Descriptor format

When stage 1 is enabled and substreams are enabled and two-level Context descriptor tables are in use (STE.S1Fmt != `0b00`), the stage 1 context pointer indicates an array of Level 1 Context descriptors which contain pointers to Level 2 CD tables.

Bits	Name	Meaning
[0]	V	0: L2Ptr invalid: 1: L2Ptr valid – CDs are indexed through to the next level table
[51:12]	L2Ptr[51:12]	Pointer to next-level table. Address bits above and below the field range are treated as zero. The programmed value must be in the range of the IAS if stage 2 is enabled for the stream causing a CD fetch through this descriptor, or in the range of the OAS if stage 2 is not enabled for the stream. See section 3.4.3 for behavior of addresses beyond IPA or PA address range. In SMMUv3.0, bits [51:48] are RES0.

If a CD fetch for a transaction with SubstreamID encounters an L1CD with V == 0, L2Ptr is IGNORED, the transaction is terminated with abort and a C_BAD_SUBSTREAMID event is recorded.

See the invalidation examples for L1STD in section 5.1. When an L1CD is changed, the non-leaf form of CMD_CFGI_CD is the minimum scope of invalidation command required to invalidate SMMU caches of the L1CD entry. Depending on the change, other CD invalidations might be required, for example:

- Changing an inactive L1CD with V == 0 to an active V == 1 form (introducing a new section of level-2 CD table) requires an invalidation of the L1CD only. Because no CDs were reachable for SubstreamIDs within the span, none require invalidation. A CMD_CFGI_CD can be used with Leaf == 0 and any SubstreamID that matches the L1CD entry.
- Changing an active L1CD with V == 1 to an inactive L1CD (decommissioning a span of SubstreamIDs) requires an invalidation of the L1CD as well as invalidation of cached CDs from the affected span. Either multiple non-leaf CMD_CFGI_CD commands, or a wider scope such as CMD_CFGI_CD_ALL, CMD_CFGI_STE or CMD_CFGI_ALL is required.

See also the CD notes in Section 5.4.1. An L1CD can be cached multiple times for the same reason as a single CD can be cached multiple times if accessed through multiple StreamIDs. This situation requires an invalidation procedure covering multiple StreamIDs.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

5.4 Context Descriptor

The CD structure contains stage 1 translation table pointers and associated fields (such as ASID and table walk attributes). Translation tables might be interpreted as AArch32 (Long/LPAE) or AArch64 formats (as controlled by the AA64 field).

Invalid or contradictory CD configurations are marked ILLEGAL. A transaction that attempts to translate through a CD containing ILLEGAL configuration is terminated with an abort and a C_BAD_CD event is recorded in the Event queue appropriate to the Security state of the transaction (as determined by SSD).

Note: In accordance with ARMv8-A [4], the interpretation of the translation table descriptor permissions is a function of the translation table format (from CD.AA64) and the Exception level to which the tables correspond (as determined by the STE StreamWorld). For example, stage 1 AArch32-format tables maintained for an EL1 StreamWorld cannot encode execute-only, whereas stage 1 AArch64-format tables maintained for an EL1 can. See [4] for more information.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

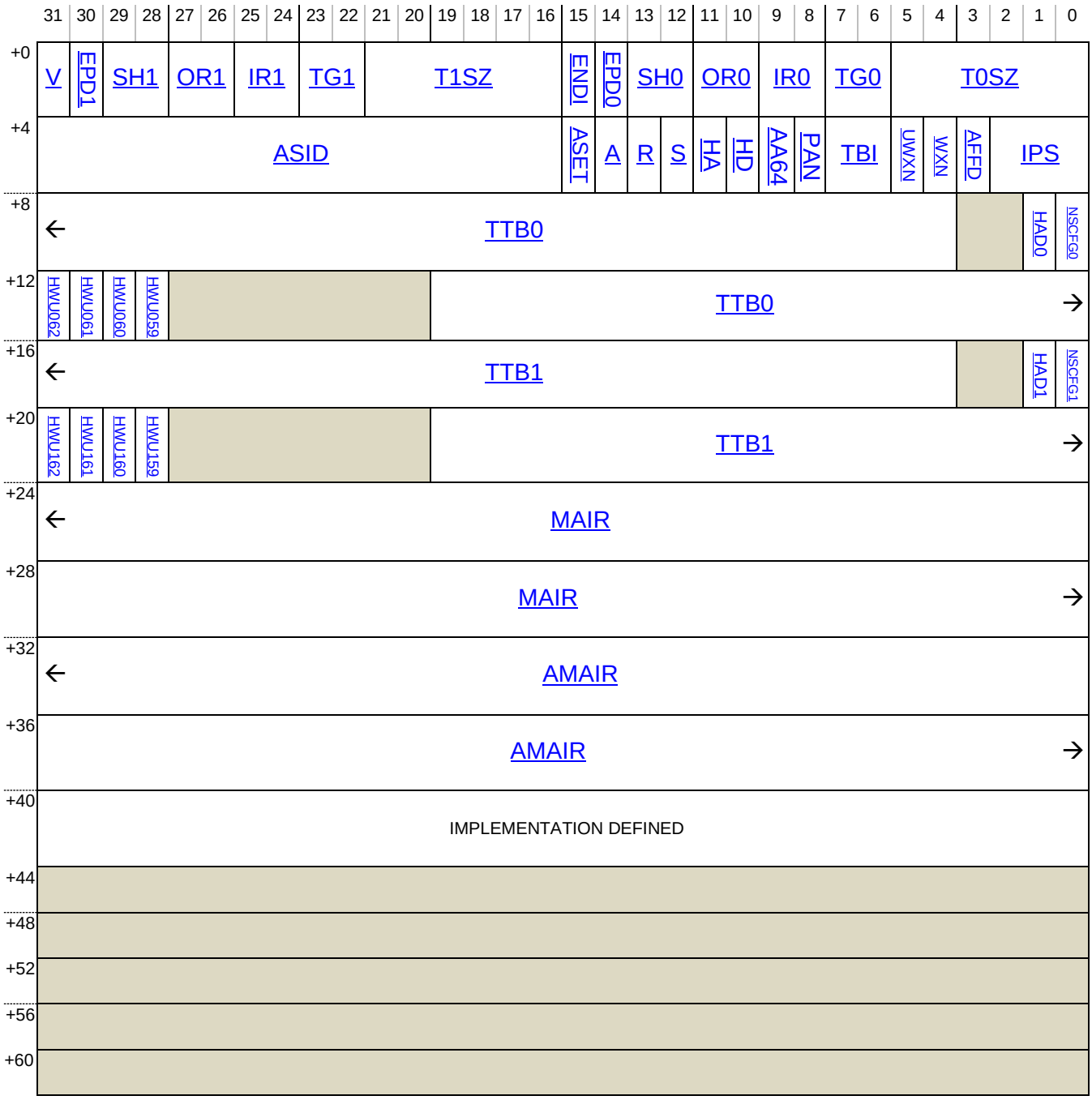


Figure 16: Context Descriptor format

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning

[5:0]	T0SZ	VA region size covered by TT0.
		
		When AA64 == 0:
		• This field is 3 bits ([2:0]) with bits [5:3]. The input region size is calculated the same as in ARMv8 AArch32 TTBCR [4].
		• The valid range of values is 0 to 7 inclusive.
		  o Note: All 3-bit values are valid, therefore it is not possible to use an out of range value when AA64 == 0.
		
		When AA64 == 1:
		• This field is 6 bits and the region size is calculated the same as in ARMv8-A AArch64 TCR_ELx [4].
		  • The maximum valid value is 39.
		  • The minimum valid value is 16 unless all of the following also hold, in which case the minimum permitted value is 12:
		  o SMMUv3.1 is supported
		  o SMMU_IDR5.VAX indicates support for 52-bit VAs
		  o The corresponding CD.TGx selects a 64KB granule.
		
		In SMMUv3.1 implementations, a CD is treated as ILLEGAL if it contains a TxSZ value outside the range of these maximum and minimum values.
		In SMMUv3.0 implementations, a fetch of a CD containing an out of range value is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:
		• The CD becomes ILLEGAL.
		• The CD is not ILLEGAL and the effective value used by the translation is 39 if the programmed value is greater than 39, or 16 if the programmed value is less than 16.
		
		IGNORED if the effective value of EPD0 == 1.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

<table>
<tr><th>Bits</th><th>Name</th><th>Meaning</th></tr>
<tr><td>[7:6]</td><td>TG0</td><td>TT0 Translation Granule size, when AA64 == 1:
<table>
<tr><td>0b00</td><td>4KB</td></tr>
<tr><td>0b01</td><td>64KB</td></tr>
<tr><td>0b10</td><td>16KB</td></tr>
<tr><td>0b11</td><td>Reserved</td></tr>
</table>
This field must only select a granule supported by the SMMU, as indicated by SMMU_IDR5. Use of an unsupported size or Reserved value is ILLEGAL, except this field is IGNORED if the effective value of EPD0 == 1.
Note: The different encoding of TG1 to TG0 is consistent with the ARMv8 Translation System [4].

When AA64 == 0 (AArch32), TG0 and TG1 are IGNORED.</td></tr>
<tr><td>[9:8]</td><td>IR0</td><td>Inner region Cacheability for TT0 access:
<table>
<tr><td>0b00</td><td>Non-cacheable</td></tr>
<tr><td>0b01</td><td>Write-back Cacheable, Read-Allocate, Write-Allocate</td></tr>
<tr><td>0b10</td><td>Write-through Cacheable, Read-Allocate</td></tr>
<tr><td>0b11</td><td>Write-back Cacheable, Read-Allocate, no Write-Allocate</td></tr>
</table>
Because the only time a translation table is written by the SMMU is when HTTU is in use, and because the atomic update might require data to be already cached, it is IMPLEMENTATION DEFINED as to whether 0b01 and 0b11 differ.

HTTU might require TTDs to be cached. It is IMPLEMENTATION DEFINED as to whether HTTU is performed when 0b00 or 0b10 are in use. ARM recommends that software use 0b01 or 0b11 when HTTU is enabled for this translation table unless the behavior of an implementation is otherwise known.

IGNORED if the effective value of EPD0 == 1.</td></tr>
<tr><td>[11:10]</td><td>OR0</td><td>Outer region Cacheability for TT0 access, as IR0.</td></tr>
</table>

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
[13:12]	SH0	Shareability for TT0 access: <table><tr><td>0b00</td><td>Non-shareable</td></tr><tr><td>0b01</td><td>Reserved (behaves as 0b00)</td></tr><tr><td>0b10</td><td>Outer Shareable</td></tr><tr><td>0b11</td><td>Inner Shareable</td></tr></table> Note: If both IR0 and OR0 == 0b00, selecting normal Non-cacheable access, the Shareability of TT0 access is taken to be OSH regardless of the value of this field. IGNORED if the effective value of EPD0 == 1.
[14]	EPD0	TT0 translation table walk disable: 0: Perform translation table walks using TT0 1: A TLB miss on an address that is translated using TT0 causes a Translation fault. No translation table walk is performed. T0SZ/TG0/IR0/OR0/SH0/TTB0 are IGNORED. Consistent with ARMv8-A translation, the EPD0 and EPD1 fields are IGNORED (and their effective value is 0) if this CD is located from an STE with StreamWorld of EL2 or EL3. It is only possible for an EL1 (Secure or Non-secure) or EL2-E2H stream to disable translation table walk using EPD0 or EPD1.
[15]	ENDI	Translation table endianness: 0: Little Endian 1: Big Endian If the effective values of both EPD0 and EPD1 are 1, this field is IGNORED. Otherwise: If SMMU_IDR0.TTENDIAN == 0b10, it is ILLEGAL to set ENDI == 1. If SMMU_IDR0.TTENDIAN == 0b11, it is ILLEGAL to set ENDI == 0.
[21:16]	T1SZ	VA region size covered by TT1, as T0SZ. IGNORED if the effective value of EPD1 == 1. This field is RES0 if StreamWorld == EL2 or EL3.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
[23:22]	TG1	TT1 Translation Granule size: `0b10` 4KB `0b11` 64KB `0b01` 16KB `0b00` Reserved This field must only select a granule supported by the SMMU, as indicated by SMMU_IDR5. Use of an unsupported size or Reserved value is ILLEGAL, except this field is IGNORED if the effective value of EPD1 == 1. Note: The different encoding of TG1 to TG0 is consistent with the ARMv8 Translation System [4]. This field is RES0 if StreamWorld == EL2 or EL3, otherwise when AA64 == 0 (AArch32), TG0 and TG1 are IGNORED.
[25:24]	IR1	Inner region Cacheability for TT1 access, as IR0. IGNORED if the effective value of EPD1 == 1. This field is RES0 if StreamWorld == EL2 or EL3.
[27:26]	OR1	Outer region Cacheability for TT1 access, as OR0. IGNORED if the effective value of EPD1 == 1. This field is RES0 if StreamWorld == EL2 or EL3.
[29:28]	SH1	Shareability for TT1 access, as SH0. IGNORED if the effective value of EPD1 == 1. This field is RES0 if StreamWorld == EL2 or EL3.
[30]	EPD1	TT1 translation table walk disable, as EPD0, but affects T1SZ/TG1/IR1/OR1/SH1/TTB1.
[31]	V	CD Valid. 0: Invalid – use of the CD by an incoming transaction is ILLEGAL 1: Valid If V == 0, the entire rest of the structure is IGNORED. An incoming transaction causing a CD with V == 0 to be located from a valid STE is terminated with an abort and a C_BAD_CD event is recorded.
[34:32]	IPS	Intermediate Physical Size: `0b000` 32 bits `0b001` 36 bits `0b010` 40 bits `0b011` 42 bits `0b100` 44 bits `0b101` 48 bits

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning

`0b110`	In SMMUv3.0 implementations, this value is Reserved and behaves as `0b101`. In SMMUv3.1 implementations, this value selects 52 bits of IPA.
`0b111`	In SMMUv3.0 implementations, this value is Reserved and behaves as `0b101`. In SMMUv3.1 implementations, this value is Reserved and behaves as `0b110`.

Software must not rely on the behavior of Reserved values.

Addresses output from the stage 1 translation table walks through either TTBx table base are range-checked against the effective value of this field, causing a stage 1 Address Size fault if out of range. See section 3.4.

When CD.AA64 == 0, IPS is IGNORED and effective Stage 1 output address size of 40 bits is always used. This reflects the 40-bit IPA used in AArch32 translations. An Address Size fault occurs if the output address bits [47:40] of an AArch32 stage 1 TTD are programmed as non-zero.

When CD.AA64 == 1:

- The effective stage 1 output address size is given by:

  ```
  Effective_IPS = MIN(CD.IPS, SMMU_IDR5.OAS);
  ```

  The effective IPS size is capped to the OAS.
- Setting this field to any value greater than the cap behaves as though this field equals the cap size. Software must not rely on this behavior.

An Address Size fault occurs if stage 1 outputs an address having bits [51:*eff_IPS*] non-zero, where *eff_IPS* is the effective number of IPA bits.

An address greater than 48 bits in size can only be output from a TTD when, in SMMUv3.1, the effective IPS is 52 and a 64KB granule is in use for that translation table. In SMMUv3.0 addresses are limited to 48 bits.

Note: If the effective IPS is 52 and a granule smaller than 64KB is used, the address output bits [51:48] are treated as `0b0000` and no Address Size fault can occur.

Note: Setting IPS == `0b110` on an implementation that supports

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
		OAS of 52 bits when a translation granule other than 64KB is in use causes no error. When a granule other than 64KB is used, the translation tables cannot encode an address greater than 48 bits and, because IPS is used to check Address Size fault, an Address Size fault resulting from an address > 48 bits cannot occur. Similarly, an Address Size fault is only possible when a granule other than 64KB is in use if the effective IPS is smaller than 48 bits.
[35]	AFFD	Access Flag Fault Disable: When HTTU is not in use because HA == 0 or HTTU is not supported, this flag determines the behavior on access of a stage 1 page whose descriptor has AF == 0: 0: An Access flag fault occurs (behavior controlled by ARS bits) 1: An Access flag fault never occurs. The TTD.AF bit is considered to be always 1. When HA == 1, this flag is IGNORED. Note: Because AFFD == 1 causes a TTD.AF bit to be considered to be 1, a resulting TLB entry will contain AF == 1.
[36]	WXN	Write eXecute Never: This flag controls overall permission of an instruction read to any writable page located using TTB{0,1}: 0: Instruction read is allowed as normal 1: Instruction read to writable page raises a Permission fault
[37]	UWXN	Unprivileged Write eXecute Never: This flag controls overall permission of a privileged instruction read to a page marked writable for user privilege that is located through TTB{0,1}: 0: Instruction read is allowed as normal 1: Instruction read from user-writable page raises a stage 1 Permission Fault In configurations for which all accesses are considered privileged, for example StreamWorld == EL2 or StreamWorld == EL3, this bit has no effect and is IGNORED. In configurations using AArch64-format stage 1 translation tables (AA64 == 1), all EL0-writable regions are treated as being PXN at EL1, the actual value of this bit is IGNORED and UWXN/PXN are effectively both 1 for these regions.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
[38]	TBI0	Top Byte Ignore for TTB0 TBIx affects generation of a Translation fault for addresses having VA[63:56] dissimilar to a sign-extension of VA[55] in the same way as TBI in an ARMv8-A. Note: Refer to section 3.9.1 for additional considerations for use of TBI with PCIe ATS.
[39]	TBI1	Top Byte Ignore for TTB1 This field is RES0 if StreamWorld == EL2 or EL3.
[40]	PAN	Privileged Access Never: When 1, this bit disables data read or data write access by privileged transactions to a virtual address where unprivileged access to the virtual address is permitted at stage 1 when the AP[2:1] == `0bx1` && APTable[0] == 0, for all APTable bits associated with that virtual address. Note: HADx removes APTable bits from this evaluation. Privileged transactions are those configured through an STE with STE.PRIVCFG == Privileged, or an STE with STE.PRIVCFG == 'Use Incoming' and marked as Privileged by the client device. This bit is IGNORED when StreamWorld == EL2 or StreamWorld == EL3 and PAN is effectively 0.
[41]	AA64	Translation table format: 0: TTB{0,1} point to AArch32-format translation tables 1: TTB{0,1} point to AArch64-format translation tables It is ILLEGAL to select AArch32 tables when either: • AArch32 tables are not supported (SMMU_IDR0.TTF[0] == 0) • StreamWorld == EL2-E2H or EL3. It is ILLEGAL to select AArch64 tables when either: • AArch64 tables are not supported (SMMU_IDR0.TTF[1] == 0) • Stage 2 translates (STE.Config[1] == 1) and STE.S2AA64 == 0. o Note: Consistent with the ARMv8-A VMSA, a 64-bit stage 1 is not supported on a 32-bit stage 2. Note: When AArch64 is selected, the IPS field selects a variable output address size, the translation granule can be altered, HTTU can be enabled and page permissions are interpreted differently, see the other fields for details.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

<table>
<tr><th>Bits</th><th>Name</th><th>Meaning</th></tr>
<tr><td>[43:42]</td><td>HA, HD</td><td>Hardware Translation Table Update of Access/Dirty flags for TT0 and TT1:
<table>
<tr><th>HA</th><th>HD</th><th></th></tr>
<tr><td>0</td><td>0</td><td>HTTU disabled</td></tr>
<tr><td>1</td><td>0</td><td>Update of Access flag enabled</td></tr>
<tr><td>0</td><td>1</td><td>Reserved – behaves as HA == 0 && HD == 0</td></tr>
<tr><td>1</td><td>1</td><td>Update of Access flag and Dirty state of the page enabled</td></tr>
</table>
These flags are IGNORED when AA64 == 0 (that is not AArch64). Otherwise:

It is ILLEGAL to set HA if SMMU_IDR0.HTTU == <code>0b00</code>.
It is ILLEGAL to set HD if SMMU_IDR0.HTTU == <code>0b00</code> or <code>0b01</code>.

Note: If HTTU is enabled when ORx or IRx indicate Non-cacheable memory, behavior is IMPLEMENTATION DEFINED, see 3.15 above. Some systems might only be able to perform atomic updates using normal cacheable memory. Incompatible attributes might cause HTTU not to be performed but system integrity must be maintained as a CD might be under control of a malicious VM.</td></tr>
<tr><td>[46:44]</td><td>{A,R,S}</td><td>Stage 1 fault behavior.

See section 5.5 below for a description of fault configuration.</td></tr>
</table>

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
[47]	ASET	ASID Set. Selects type for ASID, between sets shared and non-shared with PE ASIDs. This flag affects broadcast TLB invalidation participation and the scope within which Global TLB entries are matched: 0: ASID in shared set: This ASID, and address space described by TTB0 and TTB1, are shared with that of a process on the PE. All matching broadcast invalidation messages will invalidate TLB entries created from this context (where supported and globally enabled), keeping SMMU and PE address spaces synchronised. 1: ASID in non-shared set: TLB entries created from this context are not expected to be invalidated by some broadcast invalidations as described in section 3.17. For invalidation by ASID scope such entries require SMMU invalidation commands to be issued. This ASID represents an SMMU-local address space, not shared with PE processes, therefore broadcast invalidation from a coincidentally matching ASID is irrelevant to this address space. Use of ASET == 1 might avoid over-invalidation and improve performance. Note: All other broadcast TLB invalidations, except for those listed here, are expected to affect matching TLB entries created when ASET == 1, for example, TLBI VMALLE1IS must invalidate all TLB entries matching a given VMID, regardless of ASET. ASET must be included in any Global cached translations inserted using a CD reached through an STE with StreamWorld == NS-EL1 or Secure or EL2-E2H}. In these StreamWorlds, ASET is also intended (but not required) to be included in non-Global translations to allow them to opt-out of broadcast invalidation. Changes to a the ASET of a CD or the ASID, or both, are not automatically reflected in cached translations. ARM recommends that these are subjected to separate TLB maintenance. ASET is permitted to be included in cached translations inserted using a CD reached through an STE with StreamWorld == EL2 or EL3.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
[63:48]	ASID	Address Space Identifier: See ASET field. Tag for TLB entries inserted due to translations from this CD, differentiating them from translations with the same VA from different address spaces. ASID must tag all cached translations inserted using this CD through an STE with StreamWorld == NS-EL1 or Secure or EL2-E2H. This field is IGNORED if StreamWorld == EL2 or EL3. Otherwise, when an implementation supports only 8-bit ASIDs (SMMU_IDR0.ASID16 == 0), it is ILLEGAL for ASID[15:8] to be non-zero. If 16-bit ASIDs are supported by an implementation, the full ASID[15:0] value is used regardless of AA64. ARM expects that legacy/AArch32 software using 8-bit ASIDs will write zero-extended 8-bit values in the ASID field in this case.
[64]	NSCFG0	Non-secure attribute for the memory associated with the starting-level translation table to which TTB0 points: 0: Starting-level descriptor of TTB0 is fetched using NS == 0 access 1: Starting-level descriptor of TTB0 is fetched using NS == 1 access This field is used only when the CD is reached from a Secure STE and is otherwise IGNORED.
[65]	HAD0	Hierarchical Attribute Disable for the TTB0 region: 0: Hierarchical attributes are enabled 1: Hierarchical attributes are disabled The presence of this feature can be determined from SMMU_IDR3.HAD; when SMMU_IDR3.HAD == 0, the HAD0 and HAD1 fields are IGNORED. Otherwise: When this field is 1, the APTable, PXNTable and XNTable/UXNTable fields of table descriptors walked through TTB0 become IGNORED, might be used by software for any purpose and do not affect translation. When StreamWorld == EL2 or EL3, this effect includes the APTable[0] and PXNTable bits which are otherwise Reserved by VMSA in these translation regimes. HAD0 and HAD1 are supported for both AArch32 and AArch64 translation tables.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
[115:68]	TTB0[51:4]	Address of TT0 base. Address bits above and below the field range are implied as zero. Bits [(*x*-1):0] are treated as if all the bits are zero, where *x* is defined by the required alignment of the translation table as given by the ARMv8-A VMSA. Note: The SMMU effectively aligns the value in this field before use. A 64-byte minimum alignment on starting-level translation table addresses is imposed when TG0 selects 64KB granules and the effective IPS value indicates 52-bit output. In this case bits [5:0] are treated as zero. If the effective value of EPD0 == 1, TTB0 is IGNORED. Otherwise, it is ILLEGAL for the address in TTB0 to be outside the range described by the CD's effective IPS value. In addition, it is ILLEGAL for the address in TTB0 to be outside of a 48-bit range when TG0 selects a granule smaller than 64KB. Note: Where the effective IPS < 52, an address greater than the effective IPS that is still within the defined range of this field (up to bit [51]) causes an ILLEGAL CD. Additionally, a 52-bit address can only be used in a TTBx when a 64KB granule is configured for the TTBx. In SMMUv3.0, CD[115:112] are RES0.
[124]	HWU059	When SMMU_IDR3.PBHA == 1 and CD.HAD0 == 1, this bit controls the interpretation of bit [59] of the stage 1 translation table final-level (page or block) descriptor pointed at by CD.TTB0: • 0: Bit [59] is not interpreted by hardware for an IMPLEMENTATION DEFINED purpose. • 1: Bit [59] has IMPLEMENTATION DEFINED hardware use. This bit is IGNORED when PBHA are not supported (SMMU_IDR3.PBHA == 0) or when CD.HAD0 == 0. This bit is RES0 in SMMUv3.0.
[125]	HWU060	Similar to HWU059, but affecting descriptor bit [60].
[126]	HWU061	Similar to HWU059, but affecting descriptor bit [61].
[127]	HWU062	Similar to HWU059, but affecting descriptor bit [62].
[128]	NSCFG1	Non-secure attribute for the memory associated with the starting-level translation table to which TTB1 points. See NSCFG0. This field is RES0 if StreamWorld == EL2 or EL3.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits	Name	Meaning
[129]	HAD1	Hierarchical Attribute Disable for the TTB1 region: 0: Hierarchical attributes are enabled 1: Hierarchical attributes are disabled When SMMU_IDR3.HAD == 1, this field is RES0 if StreamWorld == EL2 or EL3.
[179:132]	TTB1[51:4]	Address of TT1 base. Address bits above and below the field range are implied as zero. See the notes below about when TTB1 might be valid. Bits [(*x*-1):0] are treated as if all the bits are zero, where *x* is defined by the required alignment of the translation table as given by the ARMv8-A VMSA [4]. Note: The SMMU effectively aligns the value in this field before use. A 64-byte minimum alignment on starting-level translation table addresses is imposed when TG1 selects 64KB granules and the effective IPS value indicates 52-bit output; bits [5:0] are treated as zero. This field is RES0 if StreamWorld == EL2 or EL3, otherwise if the effective value of EPD1 == 1, TTB1 is IGNORED. Otherwise, it is ILLEGAL for the address in TTB1 to be outside the range described by the effective IPS value of the CD. In addition, it is ILLEGAL for the address in TTB1 to be outside of a 48-bit range when TG1 selects a granule smaller than 64KB. In SMMUv3.0, CD[179:176] are RES0.
[188]	HWU159	When SMMU_IDR3.PBHA == 1 and CD.HAD1 == 1, this bit controls the interpretation of bit [59] of the Stage 1 translation table final-level (page or block) descriptor pointed at by CD.TTB1: • 0: Bit [59] is not interpreted by hardware for an IMPLEMENTATION DEFINED purpose. • 1: Bit [59] has IMPLEMENTATION DEFINED hardware usage. This bit is IGNORED when PBHA are not supported (SMMU_IDR3.PBHA == 0) or when CD.HAD1 == 0. This bit is RES0 in SMMUv3.0.
[189]	HWU160	Similar to HWU159, but affecting descriptor bit [60].
[190]	HWU161	Similar to HWU159, but affecting descriptor bit [61].
[191]	HWU162	Similar to HWU159, but affecting descriptor bit [62].
[223:192]	MAIR0[31:0]	Equivalent to ARMv8-A Memory Attribute Indirection Registers, these

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

<table>
<tr><th>Bits</th><th>Name</th><th>Meaning</th></tr>
<tr><td>[255:224]</td><td>MAIR1[31:0]</td><td>fields are indexed by AttrIndx in descriptors fetched from TT0 & TT1 and contain attributes encoded in the same way as in ARM VMSAv8-64 MAIR registers with the following exceptions:

All encodings defined as UNPREDICTABLE in VMSAv8-64 are Reserved in this architecture, and behave as follows:
<table>
<tr><th>Attr<n>[3:0]</th><th>If Attr<n>[7:4] == 0b0000</th><th>If Attr<n>[7:4] != 0b0000</th></tr>
<tr><td>0b0000</td><td>[same as VMSAv8-64]</td><td>Reserved, behaves as
Normal-iWT Transient with RW == 0b11</td></tr>
<tr><td>0b00RW where RW != 0b00</td><td>Reserved, behaves as Device-nGnRnE</td><td rowspan="4">[same as VMSAv8-64]</td></tr>
<tr><td>0b01RW where RW != 0b00</td><td>Reserved, behaves as Device-nGnRE</td></tr>
<tr><td>0b10RW where RW != 0b00</td><td>Reserved, behaves as Device-nGRE</td></tr>
<tr><td>0b11RW where RW != 0b00</td><td>Reserved, behaves as Device-GRE</td></tr>
</table></td></tr>
<tr><td>[287:256]</td><td>AMAIR0[31:0]</td><td rowspan="2">Equivalent to PE Auxiliary Memory Attribute Indirection Registers. In a typical implementation, these fields are split into eight one-byte fields corresponding to the MAIRn.Attr[n] fields, but this is not required. The content of these fields is IMPLEMENTATION DEFINED and might enable extended functionality.
If software has no implementation-specific knowledge it must set the AMAIR fields to zero. An implementation must remain compatible with generic driver software by maintaining correct behavior when the AMAIR is set to zero.</td></tr>
<tr><td>[319:288]</td><td>AMAIR1[31:0]</td></tr>
<tr><td>[351:320]</td><td>IMPLEMENTATION DEFINED</td><td>IMPLEMENTATION DEFINED, for example QoS, allocation, or transient hints for translation table access.</td></tr>
</table>

Note: Consistent with ARMv8.1 [5] PE translation, translation table descriptor bits APTable, PXNTable and UXNTable control hierarchical permission attributes for lower levels of translation table walks. Not all systems use this feature, so the CD Hierarchical Attribute Disable flags allow translation table walks using TTB0 and TTB1 to be performed ignoring these bits. Software is then free to use the APTable, PXNTable and UXNTable fields for other purposes.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

5.4.1 CD notes

When a CD is used from a stream configured with StreamWorld == EL2 or EL3 (but not EL2-E2H):

- Only one translation table is supported (TTB0) and TTB1 is unreachable.
- ASID is IGNORED.

When TTB1 is unreachable:

- The following fields become RES0:
TTB1, TBI[1], TG1, SH1, OR1, IR1, T1SZ, HAD1, NSCFG1.
- T0SZ must cover the required VA input space

A CD must only be configured to be located through a set of multiple STEs when all STEs in the set configure an identical Exception level (given by the STE StreamWorld).

Note: For example, one STE configuring a stream as EL2 and another STE configuring a stream as NS-EL1 must not both share the same CD as, in this configuration, TTB1 is both enabled and unused depending on the StreamID that locates the CD. Similarly, a CD with AA64 == 0 is ILLEGAL if reached from an STE with StreamWorld == EL2-E2H but not if reached from an STE with StreamWorld == EL1.

The legality of a CD is, in part, affected by following properties of the STE used to locate the CD:

- StreamWorld.
- S1STALLD.
- AA64 and Config[1] (to detect AArch32 stage 2 translation).

When a CD is reached from an STE whose state causes the CD to be considered ILLEGAL, C_BAD_CD is raised and the transaction causing the configuration lookup is terminated with an abort.

The interpretation of the lower bit of the translation table descriptor AP[2:1] field changes depending on the translation regime under which the table is walked, Translation tables located from STEs with StreamWorld == EL2 or EL3 are under EL2 or EL3 AArch64 ownership and, consistent with ARMv8-A PEs, the AP[1] bit (of AP[2:1]) is ignored and treated as 1, see section 13.4.1.

Depending on the presentation of an address to the SMMU, TTB1 might never be selected in some implementations, see section 3.4.1 for details.

In ARMv8-A, an inconsistency between TTBRx and IPS is reported as an Address Size fault. A CD TTBx address outside the range of IPS makes the CD ILLEGAL, recording a C_BAD_CD error.

A CD or L1CD that is successfully fetched might be cached in any state, therefore any modification, commissioning or decommissioning of a CD must be followed by a CMD_CFGI_CD command to make the

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

modification visible to the SMMU. A failed fetch (F_CD_FETCH, or a stage 2 fault where CLASS == CD) does not cause a CD or L1CD to be cached.

The translation table walks performed from TTB0 or TTB1 are always performed in IPA space if stage 2 translations are enabled (STE.Config[1] == 1). If stage 2 translations are not enabled (or if the SMMU does not implement stage 2), page table walks are always performed in PA space. CDs are fetched from IPA space if stage 2 translations are enabled, otherwise they are fetched from PA space.

Note: This enables a hypervisor to point directly to guest-managed CD structures and translation tables.

The following CD fields are permitted to be cached as part of a translation or TLB entry, and alteration requires invalidation of any TLB entry that might have cached these fields, in addition to CD structure cache invalidation:

- HAD{0,1}.
- AFFD.
- ASID+ASET (affect tagging of TLB entries, so a change may not require old entries to be invalidated).
- MAIR.
- AMAIR.
- EPD{0,1}.
- TTB{0,1}.
- T{0,1}SZ.
- OR{0,1}.
- IR{0,1}.
- SH{0,1}.
- ENDI.
- TG{0,1}.
- HA, HD.
- WXN, UWXN.
- AA64.
- TBI.
- IPS.
- NSCFG{0,1}.
- PAN.

Alteration of the remaining fields of the CD does not require an explicit invalidation of any structure other than the CD itself:

- A,R,S

Note: Changes to IMPLEMENTATION DEFINED fields might have IMPLEMENTATION DEFINED invalidation requirements.

When cached in an SMMU, a CD is uniquely identified by the tuple {StreamID, SubstreamID} including SEC_SID, when security is supported. This means that:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Multiple CDs are distinguished by StreamID, so CDs located from the same SubstreamID index but through different StreamIDs are considered separate and can contain different configuration:
    - Note: Different devices (StreamIDs) might therefore use the same SubstreamID value to associate transactions with different process address spaces in an OS. The SubstreamID namespace is local to the StreamID.
- A common CD (or CD table) shared by different STEs is permitted to be cached as multiple entries that are specific to each possible {StreamID, SubstreamID} combination.
    - A change to such a CD requires invalidation of the CD cache. When CMD_CFGI_CD(_ALL) is used, it must be issued multiple times for every {StreamID, SubstreamID} combination that the CD is reachable from.
    - Note: The SMMU might prefetch a reachable structure, so even if a CD was not accessed by a transaction with a particular StreamID, it might have been prefetched through the STE of that stream, so must still be invalidated using that StreamID.

It might arise that multiple CDs represent the same address space (therefore TLB entries), as identified by a combination of Security state, StreamWorld or translation regime, VMID tag (if relevant) and ASID tag (if relevant). This can happen in any of the following (non-exhaustive list of) cases:

- Several STEs with StreamWorld == Secure each point to their own CD that contains a common ASID value
    - Secure translation regime differentiates address spaces by ASID (but not VMID).
- Several STEs with StreamWorld == NS-EL1 contain a common VMID. Each STE points to its own CD that contains a common ASID value:
    - NS-EL1 differentiates address spaces by ASID and VMID.
- An STE with StreamWorld == EL2 points to a table of more than one CD.
    - EL2 has no ASIDs, therefore multiple CDs represent the same (single) set of translations.
- An STE with StreamWorld == EL2-E2H points to a CD table where some entries use the same ASID value:
    - EL2-E2H differentiates address spaces by ASID only.

If multiple CDs exist in the same translation regime representing the same address space, any of these CDs can insert TLB entries matching lookup in that address space. All such CDs are considered interchangeable by the SMMU and must contain identical configuration for fields that are permitted to be cacheable as part of a TLB entry.

Note: The EL2 and EL3 translation regimes do not differentiate address spaces by ASID, therefore all CDs referenced from STEs having StreamWorld EL2 or EL3 must be identical for fields permitted to be cacheable in a TLB. This includes tables of multiple CDs referenced from one STE, or CDs referenced one to one from an STE. It is not expected that SubstreamIDs (therefore a CD table with multiple entries) will be used with STEs configured for EL2 or EL3 StreamWorlds.

If a software error causes two CDs representing the same address space to differ, the result of a TLB lookup for that address space is UNPREDICTABLE. The SMMU must not allow such an error to provide device access to memory locations outside of the Security state of the stream, or that would not otherwise have been accessible given a stage 2 configuration, if present.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: Fields permitted to be cached as part of a TLB entry modify the properties of the TLB entry. A difference in CD configuration can cause TLB entries to be cached with different properties in the same address space. It would be UNPREDICTABLE as to which CD inserted a given TLB entry, therefore the properties returned by a general TLB lookup become UNPREDICTABLE.

5.4.1.1 EPDx behavior

The CD EPD0 and EPD1 fields disable translation configuration related to TTB0 and TTB1, respectively. Validity checks are not performed on fields that are disabled or IGNORED by CD.EPDx.

This table shows interaction of CD.TxSZ with EPDx and incoming addresses that select TTB0 or TTB1 translation configuration:

Virtual Address (AArch64 example shown)	Effective EPD0	Effective EPD1	T0SZ	T1SZ	Result
`0xFFFFXXXXXXXXXXXX`	0	0	Valid	Valid	Translates through TTB1
`0x0000XXXXXXXXXXXX`	0	0	Valid	Valid	Translates through TTB0
`0xFFFFXXXXXXXXXXXX`	0	0	X	Invalid[(1)]	C_BAD_CD
`0x0000XXXXXXXXXXXX`	0	0	X	Invalid[(1)]	C_BAD_CD
`0xFFFFXXXXXXXXXXXX`	0	0	Invalid[(1)]	X	C_BAD_CD
`0x0000XXXXXXXXXXXX`	0	0	Invalid[(1)]	X	C_BAD_CD
`0xFFFFXXXXXXXXXXXX`	0	1	Valid	X	TLB miss causes F_TRANSLATION (translation through TTB1 is disabled)
`0x0000XXXXXXXXXXXX`	0	1	Valid	X	Translates through TTB0
`0xFFFFXXXXXXXXXXXX`	0	1	Invalid[(1)]	X	C_BAD_CD
`0x0000XXXXXXXXXXXX`	0	1	Invalid[(1)]	X	C_BAD_CD
`0xFFFFXXXXXXXXXXXX`	1	0	X	Valid	Translates through TTB1
`0x0000XXXXXXXXXXXX`	1	0	X	Valid	TLB miss causes F_TRANSLATION (translation through TTB0 is disabled)
`0xFFFFXXXXXXXXXXXX`	1	0	X	Invalid[(1)]	C_BAD_CD
`0x0000XXXXXXXXXXXX`	1	0	X	Invalid[(1)]	C_BAD_CD

- The high-order bits or bits of the VA determine whether TTB0 or TTB1 is selected, according to the ARMv8-A VMSA.
- In the cases marked [(1)], TxSZ is shown as being an invalid value and the SMMU treating this as making the CD ILLEGAL, for the purposes of illustration. See CD.T0SZ, this is one of the CONSTRAINED UNPREDICTABLE behaviors for an out-of-range TxSZ.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- EPDx == 1 causes TxSZ to be IGNORED and an invalid TxSZ value (or an invalid TTBx or TGx value) does not lead to C_BAD_CD.
- When EPDx == 1, a translation table walk through TTBx causes F_TRANSLATION.
    - Note: The ARMv8-A VMSA allows a TLB hit to occur for an input address associated with an EPD bit set to 1, but the translation table walk is disabled upon miss.

5.4.2 Validity of CD

The following expression indicates whether a CD is considered valid or ILLEGAL, for the purposes of determining a configuration error (C_BAD_CD). Subsequent checks must be performed as part of the translation process and further faults might arise, but these are unrelated to the validity of configuration structures.

```
IPA_RANGE   =     (CD.AA64 == 0 /* LPAE */) ?
                        40 : MIN(VALUE(CD.IPS), VALUE(SMMU_IDR5.OAS)));
                  // IPA range is 40 bits for LPAE tables, else effective IPS size
                  // Note: The StreamWorld is a function of STE.STRW combined with
                  // other properties, e.g. SMMU_CR2.E2H, STE_SECURE, etc.  See the
                  // STE.STRW description for more information on StreamWorld.
                  // This expression assumes that SMMU_IDR0.HYP == 1 and
                  // SMMU_IDR0.E2H == 1;
                  // it does not illustrate the cases where STE.STRW is
                  // IGNORED or RES0 and an effective value is used (e.g. where
                  // SMMU_IDR0.S2P == 0 so the effective StreamWorld is NS-EL1):
STE_STREAMWORLD = (STE.STRW == 0b00 && !STE_SECURE) ? NS_EL1 :
                  (STE.STRW == 0b10 && !STE_SECURE && SMMU_CR2.E2H == 1) ? EL2-E2H :
                  (STE.STRW == 0b10 && !STE_SECURE && SMMU_CR2.E2H == 0) ? EL2 :
                  (STE.STRW == 0b00 && STE_SECURE) ? Secure :
                  (STE.STRW == 0b01 && STE_SECURE) ? EL3 :
                  RESERVED_STRW_VALUE_USED; // Would have caused STE to be ILLEGAL.

N_TRANSL_CFG0 =   (STE_STREAMWORLD == EL2 || STE_STREAMWORLD == EL3) ? 0 : CD.EPD0;
N_TRANSL_CFG1 =   (STE_STREAMWORLD == EL2 || STE_STREAMWORLD == EL3) ? 1 : CD.EPD1;

// The EFFECTIVE_SMMU_IDR0_STALL_MODEL is defined in section 5.2.1.

CD_ILLEGAL  =     !CD.V ||
                  (STE.S1STALLD == 1 && CD.S == 1) ||
                  (SMMU_IDR0.TERM_MODEL == 1 && CD.A == 0) ||
                  (!STE_SECURE && EFFECTIVE_SMMU_IDR0_STALL_MODEL == 0b01 &&
                        CD.S == 1) ||
                  (!STE_SECURE && EFFECTIVE_SMMU_IDR0_STALL_MODEL == 0b10 &&
                        CD.S == 0) ||
                  (STE_SECURE && SMMU_S_IDR0.STALL_MODEL == 0b01 && CD.S == 1) ||
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

```
(STE_SECURE && SMMU_S_IDR0.STALL_MODEL == 0b10 && CD.S == 0) ||
((!N_TRANSL_CFG0 || !N_TRANSL_CFG1) &&
      ((SMMU_IDR0.TTENDIAN == 0b10 && CD.ENDI == 1) ||
       (SMMU_IDR0.TTENDIAN == 0b11 && CD.ENDI == 0))) ||
(CD.AA64 == 0 && ((SMMU_IDR0.TTF[0] == 0) ||
      (STE_STREAMWORLD == EL3) ||
      (STE_STREAMWORLD == EL2-E2H))) ||
(CD.AA64 == 1 && ((SMMU_IDR0.TTF[1] == 0) ||
      (STE.Config[1] == 1 && STE.S2AA64 == 0))) ||
(CD.AA64 == 1 &&
      (((CD.HA || CD.HD) && (SMMU_IDR0.HTTU == 0b00)) ||
      (CD.HD && SMMU_IDR0.HTTU == 0b01))) ||
((STE_STREAMWORLD == NS-EL1 || STE_STREAMWORLD == Secure ||
      STE_STREAMWORLD == EL2-E2H) &&
      SMMU_IDR0.ASID16 == 0 && CD.ASID[15:8] != 0x00) ||
((CD.AA64 == 1 && CONSTR_UNPRED_TXSZ_OOR_ILLEGAL == YES) &&
      ((N_TRANSL_CFG0 == 0 && TXSZ_OUTSIDE_VALID_RANGE(CD.T0SZ))
||
       (N_TRANSL_CFG1 == 0 && TXSZ_OUTSIDE_VALID_RANGE(CD.T1SZ))))
||
(!N_TRANSL_CFG0 &&
      ((CD.AA64 == 1 && !GRANULE_SUPPORTED(CD.TG0, SMMU_IDR5)) ||
      ADDR_OUTSIDE_RANGE(CD.TTB0
            MIN(IPA_RANGE, (CD.TG0 == TG0_64KB) ? 52 : 48))) ||
(!N_TRANSL_CFG1 &&
      ((CD.AA64 == 1 && !GRANULE_SUPPORTED(CD.TG1, SMMU_IDR5)) ||
      ADDR_OUTSIDE_RANGE(CD.TTB1,
            MIN(IPA_RANGE, (CD.TG1 == TG1_64KB) ? 52 : 48))));
```

5.5 Fault configuration (A,R,S bits)

The STE contains fault configuration for faults derived from stage 2 translations (S2R, S2S) and the CD contains fault configuration for faults derived from stage 1 translations (A, R, S).

A applies to CD.A, R applies to STE.S2R and CD.R, S applies to STE.S2S and CD.S.

The A, R and S flags control fault behavior for transactions experiencing the following Translation-related faults during stage1 or stage 2 translation, see section 3.12:

- F_TRANSLATION.
- F_ACCESS.
- F_ADDR_SIZE.
- F_PERMISSION.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Transactions are terminated with an abort if they encounter any other fault or configuration error and attempt to record an event in the relevant Event queue. This case is unaffected by the A, R and S flags.

When a transaction encounters one of the four Translation-related faults, it might be immediately terminated or stalled (in which case it is later retried, or terminated, according to software command).

The following flags are used to configure fault behavior:

A	Abort behavior upon transaction termination	When set and a transaction experiencing one of the faults listed in this section is terminated, an abort or bus error is returned to the client device. When clear, such a transaction is completed successfully with RAZ/WI behavior so that the client does not receive an error condition. This configuration exists only for stage 1. The termination behavior of stage 2 is abort, as though an implied A == 1. An SMMU might only implement abort termination (with no RAZ/WI support) and indicate this behavior through the SMMU_IDR0.TERM_MODEL flag. For these implementations, configuring CD.A == 0 renders the CD ILLEGAL. The A flag has no effect on faults arising from ATS Translation Requests, which do not support RAZ/WI behavior.
R	Record event	When set, the detection of the fault is recorded in the event log. When clear, the fault record is suppressed. This field only suppresses a fault record if S == 0 and if the fault is of one of the four Translation-related faults.
S	Stall upon fault	When set, an incoming transaction experiencing one of the four Translation-related faults is stalled. No response is given to the transaction until software issues a resume or terminate command, whereupon the transaction is either retried or terminated. The stall is always reported and the R bit is ignored. Configuration or faults not arising during stage 1 or stage 2 translation cause the transaction to be terminated, regardless of the setting of this bit. When clear, an incoming transaction experiencing a fault is terminated immediately in a manner according to A. The CD.S flag has no effect on Translation-related faults arising from ATS Translation Requests, for which a R == W == 0 response is given, see section 3.9.1. Some implementations might not support stalling of eligible transactions and immediately terminate the transaction (with behavior determined by the A bit). Other implementations might not support immediate termination of transactions (that fault in a manner eligible to stall). Such implementations indicate these behaviors through the SMMU_(S_)IDR0.STALL_MODEL field.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When stage 1 translation is enabled (STE.Config[0] == 1), an STE is considered ILLEGAL if:

- SMMU_(S_)IDR0.STALL_MODEL != `0b00` and STE.S1STALLD == 1.

A CD (Stage 1 translation enabled) is considered ILLEGAL if one of the following applies:

- SMMU_(S_)IDR0.STALL_MODEL == `0b00` and STE.S1STALLD == 1 and CD.S == 1.
- SMMU_(S_)IDR0.STALL_MODEL == `0b01` and CD.S == 1.
- SMMU_(S_)IDR0.STALL_MODEL == `0b10` and CD.S == 0.

When stage 2 translation is enabled (STE.Config[1] == 1), an STE is considered ILLEGAL if one of the following applies:

- SMMU_IDR0.STALL_MODEL == `0b01` and STE.S2S == 1.
- SMMU_IDR0.STALL_MODEL == `0b10` and STE.S2S == 0.
- Note: Stage 2-enabled STEs cannot also be Secure.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

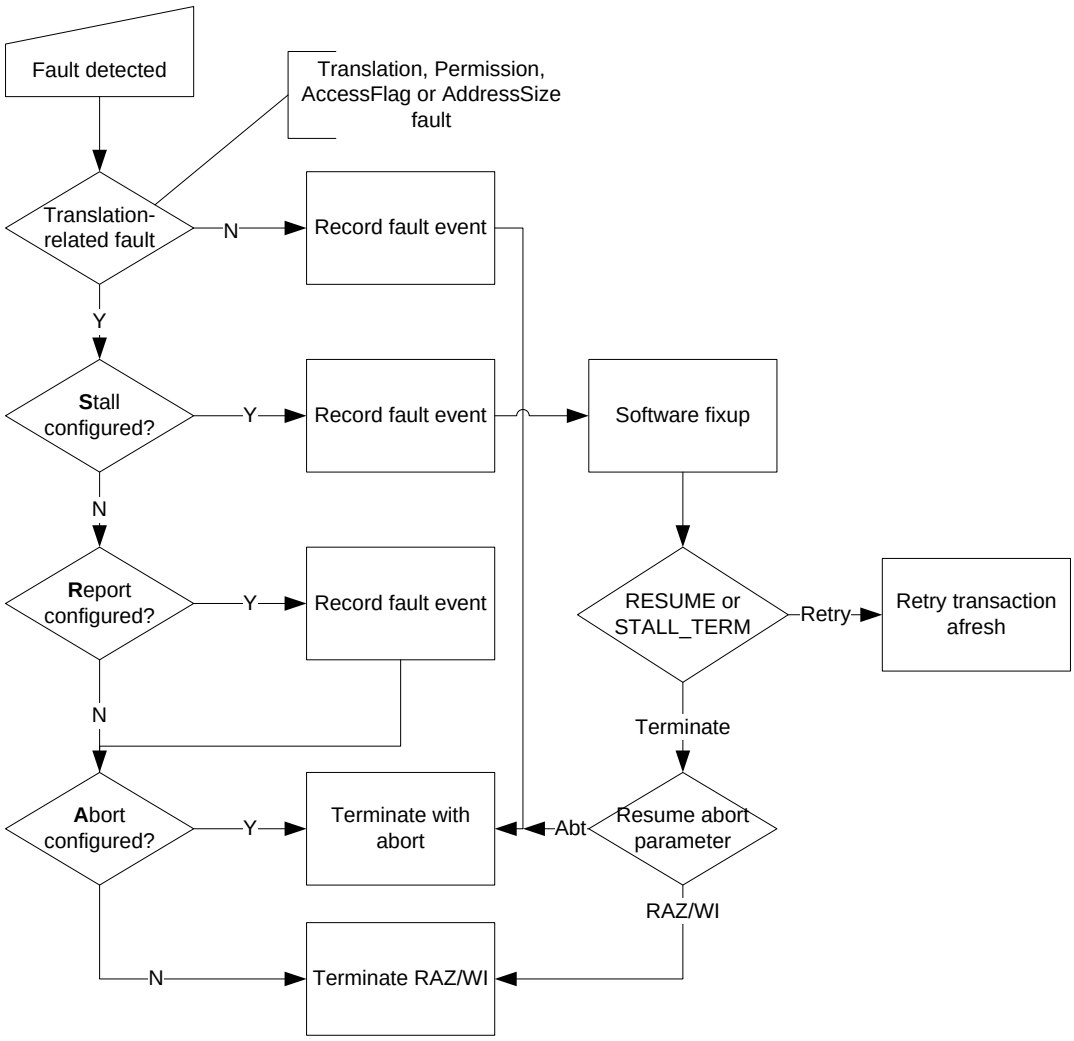


Figure 17: Fault configuration flow

These flags can be used, in the following combinations, to affect transactions experiencing one of the four Translation-related faults:

ARS	
`0b000`	Silently terminate transactions at the SMMU, with writes acknowledged successfully and reads returning zero, RAZ/WI.
`0b010`	Terminate transactions at the SMMU in a RAZ/WI manner, but record fault event as well.
`0bxx1`	Stall transaction and record fault event. In this case, the effective value of R is assumed to be 1 so that it is impossible for transactions to stall without an event being recorded. Only the stalled transaction is held and subsequent non-faulting transactions in the same stream or substream might complete (subject to interconnect ordering rules) before the stall is resolved, see section 3.12.2.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

ARS	
	Later receipt of a CMD_RESUME or CMD_STALL_TERM ensures that the transaction will be retried or terminated (with termination behavior given by the CMD_RESUME operation).
`0b100`	Transaction abort reported to client, silent.
`0b110`	Transaction abort reported to client, fault event recorded.

Note: Although the STE has no A flag, stage 2 behavior is as though an effective A flag is fixed at 1.

Note: ARS == `0bxx1` is ILLEGAL when the SMMU does not support Stall behavior, or when an STE has explicitly disabled stage 1 stall configuration but stage 1 is configured to use it:

- A CD with CD.S == 1 is considered ILLEGAL if either of the following is true:
  - SMMU_(S_)IDR0.STALL_MODEL == `0b00` and the CD is reached through an STE with STE.S1STALLD == 1.
  - SMMU_(S_)IDR0.STALL_MODEL == `0b01`.
- A STE with STE.S2S == 1 is considered ILLEGAL if SMMU_(S_)IDR0.STALL_MODEL == `0b01`.

Note: ARS == `0bxx0` is ILLEGAL when the SMMU forces Stall behavior:

- A CD with CD.S == 0 is considered ILLEGAL if SMMU_(S_)IDR0.STALL_MODEL == `0b10`.
- A STE with stage 2 translation enabled and STE.S2S == 0 is considered ILLEGAL if SMMU_IDR0.STALL_MODEL == `0b10`.

Note: There is no separate Hit Under Previous Context Fault configuration as in prior SMMU architectures because those contain a different concept of in-register context configuration being in a stall state, whereas in SMMUv3 it is only the transaction that is stalled, separate from the configuration in memory. The treatment of stalls is therefore a per-stream or per-transaction concept rather than a per-translation context concept.

Where interconnect ordering rules allow, later transactions might overtake a stalled transaction associated with the same StreamID (and SubstreamID, if supplied with all transactions). A device might restrict ordering further if required. See section 3.12.2 for more details.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6 Memory map and registers

6.1 Memory map

SMMU registers occupy two consecutive 64KB pages starting from an at least 64KB-aligned boundary. Other optional 64KB pages might follow these pages, depending on implemented features:

- If the VATOS interface is supported, one 64KB page is present containing the VATOS registers.
- Any number of IMPLEMENTATION DEFINED pages might be present.

The presence and base addresses of all optional pages are IMPLEMENTATION DEFINED and discoverable:

- The presence of VATOS support can be determined from SMMU_IDR0.VATOS.
  - The VATOS page's base address is given by SMMU_IDR2.BA_VATOS
  - This base address is called O_VATOS hereafter.
- The presence of all IMPLEMENTATION DEFINED register pages can be determined through the Peripheral ID registers and IMPLEMENTATION DEFINED registers, for example SMMU_IDR4.

The base address of optional pages is expressed as an offset from the SMMU extension address `0x20000`, in units of a 64KB page.

An SMMU that does not support IMPLEMENTATION DEFINED register pages or VATOS will occupy a 128K span of addresses, `0x00000-0x1FFFF`.

`0x00000-0x0FFFF`	SMMU registers, Page 0
`0x10000-0x1FFFF`	SMMU registers, Page 1
`0x20000-END`	0 or more optional pages
`O_VATOS + 0x0000-0xFFFF`	VATOS interface registers

6.2 Register overview

Unless specified otherwise, all registers are 32-bit, little-endian and allow read and write (R/W) accesses.

For all pages except Page 1, undefined register locations are RAZ/WI. For Page 1, access to undefined/Reserved register locations is CONSTRAINED UNPREDICTABLE and an implementation has one of the following behaviors:

- The access has RAZ/WI behavior.
- The location has the same behavior as RES0.
- The access has the same effect as would an access to the same offset in Page 0, that is Page 0 and 1 are permitted to alias.

The equivalent Page 0 offsets of registers that are defined on Page 1 are Reserved and ARM recommends that they are not accessed. Access to these offsets is CONSTRAINED UNPREDICTABLE and (depending on the implementation of Page 1) has one of the following behaviors:

- The location aliases the corresponding register in Page 1
- The access has RAZ/WI behavior.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When SMMU_S_IDR1.SECURE_IMPL == 1, SMMU_S_* registers are RAZ/WI to Non-secure access. See section 3.11 regarding NS access to SMMU_S_INIT. All other registers are accessible to both Secure and Non-secure accesses.

When SMMU_S_IDR1.SECURE_IMPL == 0, SMMU_S_* registers are RAZ/WI.

Reserved or undefined bit positions in defined registers are RES0: they read as zero and must not be written with non-zero values.

An implementation must support aligned 32-bit word access to 32-bit registers, and to both 32-bit halves of 64-bit registers. The SMMU supports aligned 64-bit access to 64-bit registers. Support for other access sizes is IMPLEMENTATION DEFINED.

The following constitute an illegal register access:

- An access of an unsupported size.
- Unaligned accesses, that is an access that does not start on a boundary equal to the access size.

An illegal register access is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The access is RAZ/WI.
- The access is observed by the register or registers spanned by the access and:
    - Write access will write any value to this register or registers including to fields of this register or registers outside of the access.
        - When the SMMU supports two Security states, this behavior does not enable a Non-secure access to affect registers that are associated with the Secure state.
    - Read access returns an UNKNOWN value.
- The access is terminated with an abort.

A 64-bit access to two adjacent 32-bit registers is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The access is RAZ/WI.
- A read returns the value of both registers and a write updates both registers, as though two 32-bit accesses were performed in an UNPREDICTABLE order.
- One of the pair of registers is read or written and the other register is RAZ/WI, as though a single 32-bit access was performed to an UNPREDICTABLE one of the pair of registers.
- The access is terminated with an abort.

Note: ARM strongly recommends not to abort accesses to registers that might be used by software that is associated with lower exception levels on the PE, such as the SMMU_VATOS_* or PMCG registers.

It is IMPLEMENTATION DEFINED whether a 64-bit access to a 64-bit register is single-copy atomic. An SMMU implementation might internally observe the access in two 32-bit halves. An aligned 32-bit word access is single-copy atomic.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

All 64-bit fields are composed of two separately-writable 32-bit register halves, with bits[63:32] at offset +4 and [31:0] at offset +0.

Some register fields have a dependency on another register field, and are described as Guarded by the other field. A Guarded register field is not permitted to be changed by software unless the field by which it is Guarded is in a certain state (and is not in the process of being Updated to or from that state). For example, SMMU_CMDQ_BASE is Guarded by SMMU_CR0.CMDQEN so that software is only permitted to change SMMU_CMDQ_BASE if SMMU_CR0.CMDQEN == 0.

Note: Software must use an appropriate barrier to ensure initialization of Guarded register fields is visible to the SMMU before the SMMU observes a set of the field by which they are Guarded.

Registers are not required to support being the target of exclusive or atomic read-modify-write update operations.

In this document, read-only means that writes are ignored.

Offset	Register	Access	Notes
0x000	SMMU_IDR0	R/O	
0x004	SMMU_IDR1		
0x008	SMMU_IDR2		
0x00C	SMMU_IDR3		
0x010	SMMU_IDR4		
0x014	SMMU_IDR5		
0x018	SMMU_IIDR		
0x01C	SMMU_AIDR		
0x020	SMMU_CR0		
0x024	SMMU_CR0ACK	R/O	
0x028	SMMU_CR1		
0x02C	SMMU_CR2		
0x040	SMMU_STATUSR	R/O	
0x044	SMMU_GBPA		
0x048	SMMU_AGBPA		
0x050	SMMU_IRQ_CTRL		
0x054	SMMU_IRQ_CTRLACK	R/O	

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Offset	Register	Access	Notes
0x060	SMMU_GERROR	R/O	
0x064	SMMU_GERRORN		
0x068	SMMU_GERROR_IRQ_CFG0		64-bit
0x070	SMMU_GERROR_IRQ_CFG1		
0x074	SMMU_GERROR_IRQ_CFG2		
0x080	SMMU_STRTAB_BASE		64-bit
0x088	SMMU_STRTAB_BASE_CFG		
0x090	SMMU_CMDQ_BASE		64-bit
0x098	SMMU_CMDQ_PROD		
0x09C	SMMU_CMDQ_CONS		
0x0A0	SMMU_EVENTQ_BASE		64-bit
0x0A8	– Reserved –		Aliases SMMU_EVENTQ_PROD or is RAZ/WI
0x0AC	– Reserved –		Aliases SMMU_EVENTQ_CONS or is RAZ/WI
0x100A8	SMMU_EVENTQ_PROD		Page 1 – see section 3.8
0x100AC	SMMU_EVENTQ_CONS		Page 1
0x0B0	SMMU_EVENTQ_IRQ_CFG0		64-bit
0x0B8	SMMU_EVENTQ_IRQ_CFG1		
0x0BC	SMMU_EVENTQ_IRQ_CFG2		
0x0C0	SMMU_PRIQ_BASE		64-bit
0x0C8	– Reserved –		Aliases PRIQ_PROD or is RAZ/WI
0x0CC	– Reserved –		Aliases PRIQ_CONS or is RAZ/WI
0x100C8	SMMU_PRIQ_PROD		Page 1
0x100CC	SMMU_PRIQ_CONS		Page 1
0x0D0	SMMU_PRIQ_IRQ_CFG0		64-bit
0x0D8	SMMU_PRIQ_IRQ_CFG1		
0x0DC	SMMU_PRIQ_IRQ_CFG2		
0x100	SMMU_GATOS_CTRL		
0x108	SMMU_GATOS_SID		64-bit
0x110	SMMU_GATOS_ADDR		64-bit
0x118	SMMU_GATOS_PAR	R/O	64-bit

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Offset	Register	Access	Notes
0x180	SMMU_VATOS_SEL		
0xE00-0xEFF	IMPLEMENTATION DEFINED		IMPLEMENTATION DEFINED
0xFD0-0xFFC	ID_REGS	R/O	IMPLEMENTATION DEFINED identification register space
0x1000-0x3FFF	IMPLEMENTATION DEFINED		IMPLEMENTATION DEFINED
0x8000	SMMU_S_IDR0	Sec, R/O	Secure programming interface
0x8004	SMMU_S_IDR1	Sec, R/O	
0x8008	SMMU_S_IDR2	Sec, R/O	
0x800C	SMMU_S_IDR3	Sec, R/O	
0x8010	SMMU_S_IDR4	Sec, R/O	
0x8020	SMMU_S_CR0	Sec	
0x8024	SMMU_S_CR0ACK	Sec, R/O	
0x8028	SMMU_S_CR1	Sec	
0x802C	SMMU_S_CR2	Sec	
0x803C	SMMU_S_INIT	Sec	
0x8044	SMMU_S_GBPA	Sec	
0x8048	SMMU_S_AGBPA	Sec	
0x8050	SMMU_S_IRQ_CTRL	Sec	
0x8054	SMMU_S_IRQ_CTRLACK	Sec, R/O	
0x8060	SMMU_S_GERROR	Sec, R/O	
0x8064	SMMU_S_GERRORN	Sec	
0x8068	SMMU_S_GERROR_IRQ_CFG0	Sec	64-bit
0x8070	SMMU_S_GERROR_IRQ_CFG1	Sec	
0x8074	SMMU_S_GERROR_IRQ_CFG2	Sec	
0x8080	SMMU_S_STRTAB_BASE	Sec	64-bit
0x8088	SMMU_S_STRTAB_BASE_CFG	Sec	

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Offset	Register	Access	Notes
0x8090	SMMU_S_CMDQ_BASE	Sec	64-bit
0x8098	SMMU_S_CMDQ_PROD	Sec	
0x809C	SMMU_S_CMDQ_CONS	Sec	
0x80A0	SMMU_S_EVENTQ_BASE	Sec	64-bit
0x80A8	SMMU_S_EVENTQ_PROD	Sec	
0x80AC	SMMU_S_EVENTQ_CONS	Sec	
0x80B0	SMMU_S_EVENTQ_IRQ_CFG0	Sec	64-bit
0x80B8	SMMU_S_EVENTQ_IRQ_CFG1	Sec	
0x80BC	SMMU_S_EVENTQ_IRQ_CFG2	Sec	
0x8100	SMMU_S_GATOS_CTRL	Sec	
0x8108	SMMU_S_GATOS_SID	Sec	64-bit
0x8110	SMMU_S_GATOS_ADDR	Sec	64-bit
0x8118	SMMU_S_GATOS_PAR	Sec, R/O	64-bit
0x8E00-0x8EFF	IMPLEMENTATION DEFINED	Sec	IMPLEMENTATION DEFINED, secure
0x9000-0xBFFF	IMPLEMENTATION DEFINED	Sec	IMPLEMENTATION DEFINED, secure
O_VATOS + 0x0A00	SMMU_VATOS_CTRL	Optional	
O_VATOS + 0x0A08	SMMU_VATOS_SID	Optional	64-bit
O_VATOS + 0x0A10	SMMU_VATOS_ADDR	Optional	64-bit
O_VATOS + 0x0A18	SMMU_VATOS_PAR	Optional, R/O	64-bit
O_VATOS + 0xE00-0xEFF	IMPLEMENTATION DEFINED		IMPLEMENTATION DEFINED

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6.3 Register formats

6.3.1 SMMU_IDR0

Read-only.

- [31:29] Reserved, RES0.

- [28:27] ST_LEVEL Multi-level Stream table support
  - `0b00`: Linear Stream table supported.
  - `0b01`: 2-level Stream Table supported in addition to Linear Stream table.
  - `0b1x`: Reserved.

- [26] TERM_MODEL Terminate model behavior
  - `0b0`: CD.A flag determines Abort or RAZ/WI behavior of a terminated transaction.
    - The act of terminating a transaction might be configured using the CD.A flag to successfully complete the transaction with RAZ/WI behavior or abort the transaction.
  - `0b1`: Terminating a transaction with RAZ/WI behavior is not supported, CD.A must be 1.
    - This means that a terminated transaction will always be aborted.

- [25:24] STALL_MODEL Stall model support
  - `0b00`: Stall and Terminate models supported.
  - `0b01`: Stall is not supported, all faults terminate transaction and STE.S2S and CD.S must be 0.
    - CMD_RESUME and CMD_STALL_TERM are not available.
  - `0b10`: Stall is forced (all faults eligible to stall cause stall), STE.S2S and CD.S must be 1.
  - `0b11`: Reserved.
    - Note: STE.S2S must be in the states above only if stage 2 translation was enabled.
  - When SMMU_S_IDR1.SECURE_IMPL == 0, this field reports whether an implementation supports the Stall model.
    When SMMU_S_IDR1.SECURE_IMPL == 1, this field reports SMMU_S_IDR0.STALL_MODEL unless SMMU_S_IDR0.STALL_MODEL == `0b00` and SMMU_S_CR0.NSSTALLD == 1 in which case Non-secure use of Stall is prevented and this field's value is `0b01`. See section 3.12.
  - Note: When SMMU_S_IDR1.SECURE_IMPL == 1, this field is related to SMMU_S_IDR0.STALL_MODEL and might be modified by SMMU_S_CR0.NSSTALLD. See section 3.12.
  - Note: An SMMU associated with a PCI system must not have STALL_MODEL == `0b10`.

- [23] Reserved.

- [22:21] TTENDIAN Endianness support for translation table walks.
  - `0b00`: Mixed-endian: CD.ENDI and STE.S2ENDI might select either endian
  - `0b01`: Reserved
  - `0b10`: Little-endian: CD.ENDI and STE.S2ENDI must select little-endian
  - `0b11`: Big-endian: CD.ENDI and STE.S2ENDI must select big-endian

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- ARM strongly recommends that a general-purpose SMMU implementation supports mixed-endian translation table walks.

- [20] VATOS Virtual ATOS page interface supported.
  - `0b0`: Virtual ATOS page interface not supported.
  - `0b1`: Virtual ATOS page interface supported.
    - ATOS must also be supported
    - Stage 1 and stage 2 translation must also be supported.

- [19] CD2L 2-level Context descriptor table supported.
  - `0b0`: 2-level CD table not supported.
  - `0b1`: 2-level CD table supported.

- [18] VMID16 16-bit VMID supported.
  - `0b0`: 16-bit VMID not supported.
    - VMID[15:8] is RES0 in command parameters and must be zero in STE.S2VMID.
  - `0b1`: 16-bit VMID supported.
  -
  - Note: The value of this field is irrelevant to software unless SMMU_IDR0.S2P == 1.

- [17] VMW VMID wildcard-matching supported for TLB invalidation.
  - `0b0`: VMID wildcard-matching not supported for TLB invalidation.
  - `0b1`: VMID wildcard-matching supported for TLB invalidation.
  - When SMMU_IDR0.S2P == 0, this field is RES0.

- [16] PRI Page Request Interface supported.
  - `0b0`: Page Request Interface not supported.
    - All SMMU_PRI_* registers are Reserved.
  - `0b1`: Page Request Interface supported.
  - When SMMU_IDR0.ATS == 0, this field is RES0.

- [15] ATOS Address Translation Operations supported.
  - `0b0`: Address Translation Operations not supported.
    - SMMU_IDR0.VATOS is RES0 and all SMMU_(S_)GATOS_* registers are Reserved.
  - `0b1`: Address Translation Operations supported.

- [14] SEV SMMU, and system, support generation of WFE wake-up events to PE.
  - `0b0`: SMMU, and system, do not support generation of WFE wake-up events to PE.
  - `0b1`: SMMU, and system, support generation of WFE wake-up events to PE.
    - Note: WFE might be used on the PE to wait for CMD_SYNC command completion.
  - This bit must reflect the ability of the system, and SMMU implementation, to convey events to all PEs that are expected to run SMMU maintenance software.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [13] MSI Message Signalled Interrupts are supported.
  - `0b0`: The implementation supports wired interrupt notifications only.
    - The MSI fields in SMMU_EVENTQ_IRQ_CFGn, SMMU_PRIQ_IRQ_CFGn and SMMU_GERROR_IRQ_CFGn are RES0.
    -
  - `0b1`: Message Signalled Interrupts are supported
  - Note: The SMMU_PRIQ_IRQ_CFG2.LO bit is not affected by whether MSIs are implemented or not.

- [12] ASID16 16-bit ASID supported.
  - `0b0`: 16-bit ASID not supported.
    - ASID[15:8] is RES0 in command parameters and must be zero in CD.ASID.
  - `0b1`: 16-bit ASID supported.
  - Note: The value of this field is irrelevant to software unless SMMU_IDR0.S1P == 1.

- [11] NS1ATS Split-stage (stage 1-only) ATS not supported.
  - `0b0`: Split-stage (stage 1-only) ATS supported.
  - `0b1`: Split-stage (stage 1-only) ATS not supported.
    - Split-stage ATS set by STE.EATS == `0b10` is not supported. See STE.EATS.
  - RES0 when SMMU_IDR0.ATS == 0 or SMMU_IDR0.S1P == 0 or SMMU_IDR0.S2P == 0.
  - Note: The value of this field is only relevant to software when ATS and both stages of translation are supported.

- [10] ATS PCIe ATS supported by SMMU.
  - `0b0`: PCIe ATS not supported by SMMU.
  - `0b1`: PCIe ATS supported by SMMU.
  - The support provided by an implementation for ATS and PRI influences interpretation of STE.EATS, ATS and PRI-related commands and SMMU_PRIQ_* registers. It does not guarantee that client devices and intermediate components also support ATS and this must be determined separately.
- [9] Hyp Hypervisor stage 1 contexts supported.
  - This flag indicates whether TLB entries might be tagged as EL2/EL2-E2H – see STE.STRW.
  - `0b0`: Hypervisor stage 1 contexts not supported.
  - `0b1`: Hypervisor stage 1 contexts supported.
  - RES0 when SMMU_IDR0.S1P == 0 or SMMU_IDR0.S2P == 1.
  - ARM recommends the implementation of Hyp/EL2 support when SMMU_IDR0.S1P == 1 && SMMU_IDR0.S2P == 1, that is when both stages of translations are supported.

- [8] DORMHINT Dormant hint supported.
  - `0b0`: Dormant hint not supported.
  - `0b1`: Dormant hint supported.

- [7:6] HTTU[1:0] H/W translation table Access flag and Dirty state of the page updates supported.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b00`: No flag updates supported.
- `0b01`: Access flag update supported.
- `0b10`: Access flag and Dirty state of the page update supported.
- `0b11`: Reserved.
- This field reflects the ability of the system, and SMMU implementation, to support hardware update.
- Note: HTTU is a feature of an SMMU implementation, but the system design also bears upon whether HTTU can be supported. For instance, HTTU requires coherent atomic updates to translation table data which need to be supported by an external interconnect. An SMMU that internally supports HTTU but does not have requisite system support must mark HTTU as `0b00` in this field.

- [5] BTM Broadcast TLB Maintenance.
  - `0b0`: Broadcast TLB maintenance not supported.
  - `0b1`: Broadcast TLB maintenance supported.
  - This bit reflects the ability of the system, and SMMU implementation, to support broadcast maintenance. If either the SMMU, or the system, or the interconnect cannot fully support broadcast TLB maintenance, this bit reads as 0.

- [4] COHACC Coherent access supported to translations, structures and queues.
  - `0b0`: Coherent access for translations, structures and queues is not supported.
  - `0b1`: IO-coherent access is supported for:
    - Translation table walks.
    - Fetches of L1STD, STE, L1CD and CD.
    - Command queue, Event queue and PRI queue access.
    - GERROR, CMD_SYNC, Event queue and PRI queue MSIs, if supported.
    - Whether a specific access is performed in a cacheable shareable manner is dependent on the access type configured for access to structures, queues and translation table walks.
  - This bit reflects the ability of the system, and SMMU implementation, to support IO-Coherent access to memory shared coherently with the PE. If either the SMMU or system cannot fully support IO-coherent access to SMMU structures/queues/translations, this reads as 0.
  - Note: This bit only pertains to accesses made directly by the SMMU in response to internal operations. It does not indicate that transactions from client devices are also IO-coherent, this capability must be determined in a system-specific manner.
  - Note: For embedded implementations using preset tables or queues, this bit only pertains to accesses made outside of the preset structures.

- [3:2] TTF[1:0] Translation Table Formats supported at both stage 1 and stage 2.
  - `0b00`: Reserved
  - `0b01`: AArch32 (LPAE)
  - `0b10`: AArch64
  - `0b11`: AArch32 and AArch64

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [1] S1P Stage1 translation supported.
  - `0b0`: Stage1 translation not supported.
  - `0b1`: Stage 1 translation supported.

- [0] S2P Stage2 translation supported.
  - `0b0`: Stage 2 translation not supported.
  - `0b1`: Stage 2 translation supported.

6.3.2 SMMU_IDR1

Read-only.

- [31] Reserved

- [30] TABLES_PRESET Table base addresses fixed.
  - `0b0`: The contents of the registers SMMU_STRTAB_BASE and SMMU_STRTAB_BASE_CFG are not fixed. When SMMU_S_IDR1.SECURE_IMPL == 1, the contents of the registers SMMU_S_STRTAB_BASE and SMMU_S_STRTAB_BASE_CFG are not fixed.
  - `0b1`: The contents of the registers SMMU_(S_)STRTAB_BASE and SMMU_(S_)STRTAB_BASE_CFG are fixed.

- [29] QUEUES_PRESET Queue base addresses fixed.
  - `0b0`: The contents of the registers SMMU_CMDQ_BASE, SMMU_EVENTQ_BASE, and, if present, SMMU_PRIQ_BASE are not fixed. When SMMU_S_IDR1.SECURE_IMPL == 1, the contents of the registers SMMU_S_CMDQ_BASE and SMMU_S_EVENTQ_BASE are not fixed.
  - `0b1`: The contents of the registers SMMU_(S_)CMDQ_BASE, SMMU_(S_)EVENTQ_BASE, and, if present, SMMU_PRIQ_BASE, are fixed.

- [28] REL Relative base pointers
  - `0b0`: When the corresponding preset field is set, base address registers report an absolute address.
  - `0b1`: When the corresponding preset field is set, base address registers report an address offset.
    - Relative addresses are calculated using an addition of the unsigned ADDR field onto the base address of Page 0.
  - When SMMU_IDR1.TABLES_PRESET == 0 and SMMU_IDR1.QUEUES_PRESET == 0, this field is RES0.

- [27] ATTR_TYPES_OVR Incoming MemType, Shareability, allocation/transient hints override.
  - `0b0`: Incoming attributes cannot be overridden before translation or by global bypass.
  - `0b1`: Incoming attributes can be overridden.

- [26] ATTR_PERMS_OVR Incoming Data/Inst, User/Privileged, NS override.
  - `0b0`: Incoming attributes cannot be overridden before translation or by global bypass.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b1`: Incoming attributes can be overridden.

- [25:21] CMDQS[4:0] Maximum number of Command queue entries.
  - Maximum number of entries as $\log_2$(entries)
    - Maximum value 19.
  - Note: The index register values include an extra bit for wrap. Therefore, a queue with 2^N entries has indices of N bits, but an index register containing (N + 1) bits.

- [20:16] EVENTQS[4:0] Maximum number of Event queue entries.
  - Maximum number of entries as $\log_2$(entries).
    - Maximum value 19.

- [15:11] PRIQS[4:0] Maximum number of PRI queue entries.
  - Maximum number of entries as $\log_2$(entries).
    - The highest possible value of this field is 19.
    - If SMMU_IDR0.PRI == 0, this field has an implementation-specific value.

- [10:6] SSIDSIZE[4:0] Max bits of SubstreamID.
  - Valid range 0 to 20 inclusive, 0 meaning no substreams are supported.
  - Reflects physical SubstreamID representation size, that is the SMMU cannot represent or be presented with SubstreamIDs greater than SSIDSIZE.

- [5:0] SIDSIZE[5:0] Max bits of StreamID.
  - This value is between 0 and 32 inclusive.
    - Note: 0 is a legal value. In this case the SMMU supports one stream.
  - This must reflect the physical StreamID size, that is the SMMU cannot represent or be presented with StreamIDs greater than SIDSIZE.
  - When SMMU_IDR1.SIDSIZE ≥7, SMMU_IDR0.ST_LEVEL != `0b00`.

6.3.3 SMMU_IDR2

Read-only.

- [31:10] Reserved, RES0.

- [9:0] BA_VATOS VATOS page base address offset.
  - If SMMU_IDR0.VATOS == 0, no VATOS page is present and field is RES0.

All BA values are encoded as an unsigned offset from SMMU address `0x20000` in units of 64KB:

Page_Address = SMMU_BASE + `0x20000` + (BA * `0x10000`)

6.3.4 SMMU_IDR3

Read-only.

- [31:6] Reserved, RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [5] PPS
  - `0b0`: The STE.PPAR field determines whether a PASID is used on a PRI auto-generated response.
  - `0b1`: If the request had a PASID, it is used on a PRI auto-generated response for PRI queue overflow, in the same way as when STE.PPAR == 1. The STE.PPAR field is not checked, and the value is IGNORED.
  - When SMMU_IDR0.PRI == 0 or SMMU_IDR1.SSIDSIZE == 0, this field is RES0.

- [4] XNX
  - `0b0`: EL0/EL1 execute control distinction at stage 2 not supported.
  - `0b1`: EL0/EL1 execute control distinction at stage 2 supported for both AArch64 and AArch32 stage 2 translation tables.
    - This feature extends the stage 2 TTD.XN field to 2 bits which are encoded, and behave, as described in ARMv8.2.
  - In SMMUv3.0, this field is RES0.
  - IN SMMUv3.1, support for this feature is mandatory when stage 2 is supported, that is when SMMU_IDR0.S2P == 1.

- [3] PBHA Page-Based Hardware Attributes presence
  - `0b0`: Page-Based Hardware Attributes not supported.
    - SMMU_IDR3.HAD determines whether Hierarchical Attribute Disables supported.
  - `0b1`: Page-Based Hardware Attributes are supported.
    - SMMU_IDR3.HAD must be 1.
  - In SMMUv3.0, this field is RES0.

- [2] HAD Hierarchical Attribute Disable presence
  - `0b0`: No Hierarchical Attribute Disable support. CD.HAD0 and CD.HAD1 are IGNORED.
  - `0b1`: CD.HAD0 and CD.HAD1 control Hierarchical Attribute Disable.
  - In SMMUv3.0, support for this feature is optional when stage 1 is supported, that is when SMMU_IDR0.S1P == 1.
  - In SMMUv3.1, support for this feature is mandatory when stage 1 is supported, that is when SMMU_IDR0.S1P == 1.
  - When SMMU_IDR0.S1P == 0, HAD == 0.

- [1:0] Reserved, RES0.

6.3.5 SMMU_IDR4

Read-only.

- [31:0] IMPLEMENTATION DEFINED

The contents of this register are IMPLEMENTATION DEFINED and can be used to identify the presence of other IMPLEMENTATION DEFINED register regions elsewhere in the memory map.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6.3.6 SMMU_IDR5

Read-only.

- [31:16] STALL_MAX Maximum number of outstanding stalled transactions supported by SMMU and system
  - The SMMU guarantees that the total number of Stall fault records that will be recorded in any Event queue, without any having been the subject of a resume or terminate command, will not exceed this number.
  - This field is RES0 if SMMU_S_IDR1.SECURE_IMPL == 0 and SMMU_IDR0.STALL_MODEL == `0b01`, or if SMMU_S_IDR1.SECURE_IMPL == 1 and SMMU_S_IDR0.STALL_MODEL == `0b01`.

- [15:12] Reserved, RES0.

- [11:10] VAX Virtual Address eXtend
  - `0b00`: Virtual addresses of up to 48 bits can be translated per CD.TTBx.
  - `0b01`: Virtual addresses of up to 52 bits can be translated per CD.TTBx.
  - Other values reserved.
  - In SMMUv3.0, this field is RES0.
  - An implementation is permitted to support VAX == `0b01` independently of whether OAS == `0b110`.
  - If VAX == 0b01, GRAN64K must be 1.

- [9:7] Reserved, RES0.

- [6] GRAN64K 64KB translation granule supported.
  - `0b0`: 64KB translation granule not supported.
  - `0b1`: 64KB translation granule supported.

- [5] GRAN16K 16KB translation granule supported.
  - `0b0`: 16KB translation granule not supported.
  - `0b1`: 16KB translation granule supported.

- [4] GRAN4K 4KB translation granule supported.
  - `0b0`: 4KB translation granule not supported.
  - `0b1`: 4KB translation granule supported.
  - When SMMU_IDR0.TTF[0] == 1, that is when AArch32 translation table are supported, this field is RES1.

- [3] Reserved, RES0.

- [2:0] OAS Output Address Size
  - Size of physical address output from SMMU.
  - `0b000`: 32 bits
  - `0b001`: 36 bits
  - `0b010`: 40 bits

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b011`: 42 bits
- `0b100`: 44 bits
- `0b101`: 48 bits
- `0b110`: 52 bits
    - In SMMUv3.0, this value is Reserved.
- `0b111`: Reserved.
    - Any future use of reserved encodings will ensure that any new encoded size will never be smaller than those defined in this version of the architecture.
- This value must match the system physical address size, see section 3.4.
- Note: Where reference is made to OAS, it is the size value that is referenced, not the literal value of this field.
- If OAS indicates 52 bits, GRAN64K must be 1.

Note: ARM recommends that SMMUv3 implementations support at least 4K and 64K granules.

6.3.7 SMMU_IIDR

Read-only.

- [31:20] ProductID
    - IMPLEMENTATION DEFINED value identifying the SMMU part.
    - Note: Normally, when the SMMU_PIDR{0,1} registers are present, ARM expects that the SMMU_PIDR{0,1}.PART_{0,1} fields match the value of SMMU_IIDR.ProductID. If required, however, an implementation is permitted to provide values for SMMU_PIDR.{0,1}.PART_{0,1} that do not match the value of SMMU_IIDR.ProductID.
- [19:16] Variant
    - IMPLEMENTATION DEFINED value used to distinguish product variants, or major revisions of the product
- [15:12] Revision
    - IMPLEMENTATION DEFINED value used to distinguish minor revisions of the product
- [11:0] Implementer
    - Contains the JEP106 code of the company that implemented the SMMU:
    - [11:8] The JEP106 continuation code of the implementer.
    - [7] Always 0.
    - [6:0] The JEP106 identity code of the implementer.
    - For an ARM implementation, bits[11:0] are 0x43B.
    - Matches the SMMU_PIDR{1,2,4}.DES_{0,1,2} fields, if SMMU_PIDR{1,2,4} are present.

Provides information about the implementation and implementer of the SMMU, and architecture version supported.

Note: This register duplicates some of the information that might be present in the ID_REGS SMMU_CIDRx/ SMMU_PIDRx fields. However, those fields are not required to be present in all implementations, so this register provides a way for software to probe this information in a generic way. ARM expects that the

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

SMMU_CIDRx/SMMU_PIDRx fields are used by ARM CoreSight and related debug mechanisms rather than primarily being for the use of drivers.

6.3.8 SMMU_AIDR

Read-only.

- [31:8] Reserved, RES0.
- [7:4] ArchMajorRev.
- [3:0] ArchMinorRev.

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

- `0x00` == SMMUv3.0
- `0x01` == SMMUv3.1

All other values reserved.

This register identifies the SMMU architecture version to which the implementation conforms.

6.3.9 SMMU_CR0

- [31:9] Reserved.

- [8:6] VMW VMID Wildcard
  - `0b000`: TLB invalidations match VMID tags exactly.
  - `0b001`: TLB invalidations match VMID[N:1].
  - `0b010`: TLB invalidations match VMID[N:2].
  - `0b011`: TLB invalidations match VMID[N:3].
  - `0b100`: TLB invalidations match VMID[N:4].
  - All other values are reserved, and behave as `0b000`.
    - N == upper bit of VMID as determined by SMMU_IDR0.VMID16
  - This field has no effect on VMID matching on translation lookup.
  - If SMMU_IDR0.VMW == 0, this field is RES0.

- [5] Reserved, RES0.

- [4] ATSCHK ATS behavior.
  - `0b0`: Fast mode, all ATS Translated traffic passes through the SMMU without Stream table or TLB lookup.
  - `0b1`: Safe mode, all ATS Translated traffic is checked against the corresponding STE.EATS field to determine whether the StreamID is allowed to produce Translated transactions.
  - If SMMU_IDR0.ATS == 0, this field is Reserved

- [3] CMDQEN Enable Command queue processing.
  - `0b0`: Processing of commands from the Non-secure Command queue is disabled.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- o `0b1`: Processing of commands from the Non-secure Command queue is enabled.

- [2] EVENTQEN Enable Event queue writes.
  - o `0b0`: Writes to the Non-secure Event queue are disabled.
  - o `0b1`: Writes to the Non-secure Event queue are enabled.

- [1] PRIQEN Enable PRI queue writes.
  - o `0b0`: Writes to the PRI queue are disabled.
  - o `0b1`: Writes to the PRI queue are enabled.
  - o If SMMU_IDR0.PRI == 0, this field is RES0.

- [0] SMMUEN Non-secure SMMU enable
  - o `0b0`: All Non-secure streams bypass SMMU, with attributes determined from SMMU_GBPA.
  - o `0b1`: All Non-secure streams are checked against configuration structures, and might undergo translation.

All fields reset to zero.

Each field in this register has a corresponding field in SMMU_CR0ACK. An individual field is described as *Updated* after the value of the field observed in SMMU_CR0ACK matches the value that was written to the field in SMMU_CR0. Reserved fields in SMMU_CR0 are not reflected in SMMU_CR0ACK. To ensure a field change has taken effect, software must poll the equivalent field in SMMU_CR0ACK after writing the field in this register.

Each field in this register is independent and unaffected by ongoing update procedures of adjacent fields.

Update of a field must complete in finite time, but is not required to occur immediately. The Update process has side-effects which are guaranteed to be complete by the time update completes.

A field that has been written is considered to be in a transitional state until Update has completed. Any SMMU function depending on its value observes the old value until the new value takes effect at an UNPREDICTABLE point before Update completes, whereupon the new value is guaranteed to be used. Therefore:

- A new value written to a field cannot be assumed to have taken effect until Update completes.
- A new value written to a field cannot be assumed not to have taken effect immediately after the write is observed by the SMMU.

A written value is observable to reads of the register even before Update has completed.

Anywhere this document refers to behavior depending on a field value (for example, a rule of the form “REG must only be changed if SMMUEN == 0”), it is the post-Update value that is referred to. In this example, the rule would be broken were REG to be changed after the point that SMMU_(S_)SMMUEN has been written to 1 even if Update has not completed. Similarly, a field that has been written and is still in a transitional state (pre-Update completion) must be considered to still have the old value for the purposes of constraints the old value places upon software. For example, SMMU_CMDQ_CONS must not be written when CMDQEN == 1, or during an as-yet incomplete transition to 0 (as CMDQEN must still be considered to be 1).

After altering a field value, software must not alter the value of the field again before the Update of the initial alteration is complete. Behavior on doing so is CONSTRAINED UNPREDICTABLE and one of the following occurs:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The new value is stored and the Update completes with any of the values written:
    - Note: The effective field value in use might not match that read back from this register.
- The new value is ignored and Update completes using the first value (reflected in SMMU_CR0ACK).
- Cease Update if the new value is the same as the original value before the first write. Note: This means no update side-effects would occur.

Note: A write with the same value (on that is not altered) is permitted. This might occur when altering an unrelated field in the same register while an earlier field Update is in process.

6.3.9.1 VMW

Update completes after both of the following occur:

- All received broadcast TLB maintenance operations are guaranteed to behave under the new value.
- A fetched CMD_TLBI_* command specifying a VMID is guaranteed to be processed using the new value.

This field is permitted to be cached in a TLB; an Update of this field must be followed by invalidation of all NS-EL1 TLB entries.

This field must not be changed while a CMD_TLBI_* command specifying a VMID, or incoming broadcast TLB invalidation operations could be being processed. If this is done, the invalidations are not guaranteed to affect TLB entries with the specified VMIDs.

Note: ARM recommends that software stops issuing invalidation commands and uses CMD_SYNC to ensure any prior invalidation commands are complete before changing this value. Similarly, ARM recommends that software completes all relevant broadcast TLBI operations before changing this field, and avoids issuing subsequent operations until Update is complete.

6.3.9.2 ATSCHK

Update completes after both of the following occur:

- Newly-fetched configuration is guaranteed to be interpreted with the new value.
- ATS Translated Transactions are guaranteed to be treated with the new value.

This bit is permitted to be cached in configuration caches. Update of this bit must be followed by invalidation of all STEs associated with ATS traffic.

In addition, this bit must not be cleared when traffic could encounter valid STEs having EATS == `0b10`. See STE.EATS and section 3.9.2 for details of this rule. If this is done, ATS Translated transactions through such STEs have an IPA address and will be presented to the system directly instead of undergoing stage 2 translation. The point at which this begins to occur is UNPREDICTABLE.

6.3.9.3 CMDQEN

When SMMU_(S_)CR0.CMDQEN completes update from 1 to 0:

- Command processing has stopped.
- Any commands that are in progress have been Consumed in their entirety, and no new commands are fetched from the Command queue. All previous Command queue reads have completed and no reads will later become visible to the system that originated from the previous CMDQEN == 1 configuration.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Consumed commands are not guaranteed to be complete unless the last Consumed command was a CMD_SYNC (whose effects are to force completion of prior commands).
- The index after the last Consumed command or the index of the first unprocessed command, if any, is observable in SMMU_(S_)CMDQ_CONS.

When CMDQEN has completed Update to 1, the SMMU begins processing commands if the CMDQ_PROD/CMDQ_CONS indexes indicate the queue is non-empty and no Command queue error is present. See section 4.

Commands are not fetched when CMDQEN == 0.

6.3.9.4 EVENTQEN

When SMMU_(S_)CR0.EVENTQEN is transitioned from 1 to 0, SMMU accesses to the queue stop. EVENTQEN completes Update when all committed event records that are in progress become visible in the queue and any Event queue abort conditions are visible in SMMU_(S_)GERROR.EVENTQ_ABT_ERR. Uncommitted events from terminated faulting transactions are discarded when the queue becomes unwritable. See section 7.2.

6.3.9.5 PRIQEN

The effective value of PRIQEN is dependent on SMMUEN, however the value of SMMU_CR0ACK.PRIQEN is solely affected by the actual value of the SMMU_CR0.PRIQEN field.

When the effective value of SMMU_CR0.PRIQEN is transitioned from 1 to 0, page request writes into the queue stop. PRIQEN completes Update when all committed page request records that are in progress become visible in the queue. When the effective value of PRIQEN == 0, incoming requests are discarded; see section 8.2.

The SMMU_PRIQ_* registers are Guarded by the actual value of the SMMU_CR0.PRIQEN field, not the effective value.

Note: This means that clearing SMMUEN but leaving PRIQEN == 1 is not a permitted method for changing the SMMU_PRIQ_* register configuration.

6.3.9.6 SMMUEN

SMMU_CR0.SMMUEN controls translation through the Non-secure interface and behavior of transactions on Non-secure streams. When SMMU_S_IDR1.SECURE_IMPL == 1, SMMU_S_CR0.SMMUEN controls transactions on Secure streams) and the SMMU might be translating Secure transactions, even if SMMU_CR0.SMMUEN == 0. In all cases, the effect of the SMMUEN of one programming interface does not affect transactions or requests associated with the other programming interface.

When SMMU_(S_)CR0.SMMUEN == 0:

- Incoming transactions on streams with Security state matching that of the SMMUEN do not undergo translation, and their behavior is controlled by SMMU_(S_)GPBA:
    - If SMMU_(S_)GPBA.ABORT == 0, the transactions bypass the SMMU with attributes determined by the other fields in SMMU_(S_)GBPA. This includes transactions supplied with a SubstreamID.
    - If SMMU_(S_)GPBA.ABORT == 1, the transactions are terminated with abort.
- When SMMU_CR0.SMMUEN == 0, the ATS interface is not operational:
    - Incoming ATS Translation Requests are returned with Unsupported Request status.
    - CMD_ATC_INV and CMD_PRI_RESP commands are ignored.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

    - Invalidation Completion messages are ignored.
    - The effective value of PRIQEN is 0 (for SMMU_CR0.SMMUEN) and incoming PRI Page Requests are discarded.
    - All clients of the interface must undergo re-initialization when the SMMU is re-enabled. For PCIe clients, this will mean the Root Complex and endpoint ATS and PRI facilities need to undergo re-initialization.
    - ATS Translated transactions are terminated with an abort.
- Configuration or translation structures are not accessed:
    - The SMMU does not access the Stream table and ignores the contents of SMMU_(S_)STRTAB_* configuration registers, which might be written by software when in this state.
    - Translation and configuration cache entries are not inserted or modified, except for invalidation by maintenance commands or broadcast operations.
        - Note: Maintenance commands issued while SMMUEN == 0 can therefore guarantee the targeted entries do not exist in SMMU caches after the command has completed.
        - Note: The 'other' Security state might still have SMMUEN == 1 and therefore be inserting cache entries for that Security state. As these entries are not visible to or affected by the Non-secure programming interface, this is only a consideration for the Secure programming interface which can maintain Non-secure cache entries.
    - Prefetch commands do not access configuration or translations nor insert entries thereof into caches.
    - HTTU is not performed. Speculative setting of Access flag is prohibited.
- As translation does not occur for bypassing transactions, translation-related events are not recorded. See section 7.2 for events that are permitted to be recorded when SMMUEN == 0.
- ATOS translation requests are not processed, see section 6.3.37.
- Commands and events are still processed, or recorded, as controlled by CMDQEN and EVENTQEN.

Completion of an Update of SMMUEN from 0 to 1 ensures that:

- Configuration written to SMMU_(S_)CR2 has taken effect, see section 6.3.12.
- All new transactions will be treated with STE configuration relevant to their stream, and will not undergo SMMU bypass.
- All associated ATOS_CTRL.RUN fields are 0, see section 6.3.37.

Completion of an Update of SMMUEN from 1 to 0 ensures that:

- All stalled transactions that might be present and that are associated with the programming interface of the SMMUEN have been marked to be terminated with an abort and no new transactions can become stalled.
    - The STAG value of a stall event record relating to a stalled transaction affected by this update is returned to the set of values that the SMMU might use in future stall event records.
    - This transition does not guarantee that stalled transactions have already been terminated by the time of the completion. Software must wait for completion of outstanding transactions in an IMPLEMENTATION DEFINED manner.
    - Note: New transactions cannot stall because SMMU translation is disabled.
- Effective PRIQEN value has transitioned to 0, including PRIQEN side-effects.
- ATOS-specific initialization and termination has completed, see section 6.3.37 for details.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- ATOS translation requests that are underway have either completed or are terminated with a INTERNAL_ERR fault. The ATOS_CTRL.RUN fields of all affected ATOS register groups have been cleared, and ATOS_PAR has been updated with the result by the time Update completes.
- All new transactions associated with the programming interface of the SMMUEN will undergo SMMU bypass (using the SMMU_(S_)GBPA attributes).

Note: At the point of transitioning SMMUEN, there might be transactions that are in progress that are buffered in the interconnect. The SMMU has no control over these transactions and the system might provide a mechanism to ensure they are flushed before SMMUEN is cleared, if required.

The path of a transaction through the SMMU is atomic with respect to changes of SMMUEN, the SMMU treats a transaction as though SMMUEN did not change mid-way during the path of the transaction path through the SMMU, even if temporally this is not the case. Therefore, when SMMUEN changes, two groups of transactions that are progress are formed for transactions relevant to the Security state of the SMMUEN in question: those that behave according to the old SMMUEN value, and those that behave according to the new value, so that it appears as though the SMMUEN change instantaneously takes effect at some point in time and incoming transactions are processed in their entirety either before or after this point. There is no requirement for the boundary between these groups to be predictable when SMMUEN is altered in the middle of a stream of transactions. However, interconnect ordering guarantees are maintained throughout. The completion of an update to SMMUEN guarantees that all new transactions arriving at the SMMU will be treated with the new value.

A change to SMMUEN is not required to invalidate cached configuration or TLB entries.

6.3.10 SMMU_CR0ACK

Read-only.

- Same fields as SMMU_CR0.

Resets to zero. Fields in this register RAZ if their corresponding SMMU_CR0 field is Reserved.

6.3.11 SMMU_CR1

- [31:12] Reserved, RES0.
- [11:10] TABLE_SH[1:0] Table access Shareability.
  - `0b00`: Non-shareable.
  - `0b01`: Reserved, treated as `0b00`.
  - `0b10`: Outer Shareable.
  - `0b11`: Inner Shareable
    - Note: When SMMU_CR1.TABLE_OC == `0b00` and SMMU_CR1.TABLE_IC == `0b00`, this field is IGNORED and behaves as Outer Shareable.
- [9:8] TABLE_OC[1:0] Table access Outer Cacheability.
  - `0b00`: Non-cacheable.
  - `0b01`: Write-Back Cacheable.
  - `0b10`: Write-Through Cacheable.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- o `0b11`: Reserved, treated as `0b00`.

- [7:6] TABLE_IC[1:0] Table access Inner Cacheability.
  - o `0b00`: Non-cacheable.
  - o `0b01`: Write-Back Cacheable.
  - o `0b10`: Write-Through Cacheable.
  - o `0b11`: Reserved, treated as `0b00`.

- [5:4] QUEUE_SH[1:0] Queue access Shareability.
  - o `0b00`: Non-shareable.
  - o `0b01`: Reserved, treated as `0b00`.
  - o `0b10`: Outer Shareable.
  - o `0b11`: Inner Shareable.
    - When SMMU_CR1.QUEUE_OC == `0b00` and SMMU_CR1.QUEUE_IC == `0b00`, this field is IGNORED and behaves as Outer Shareability.

- [3:2] QUEUE_OC[1:0] Queue access Outer Cacheability.
  - o `0b00`: Non-cacheable.
  - o `0b01`: Write-Back Cacheable.
  - o `0b10`: Write-Through Cacheable.
  - o `0b11`: Reserved, treated as `0b00`.

- [1:0] QUEUE_IC[1:0] Queue access Inner Cacheability.
  - o `0b00`: Non-cacheable
  - o `0b01`: Write-Back Cacheable
  - o `0b10`: Write-Through Cacheable
  - o `0b11`: Reserved, treated as `0b00`.

The TABLE_* fields set the attributes for access to memory through the SMMU_STRTAB_BASE.ADDR pointer. The QUEUE_* fields set the attributes for access to memory through SMMU_CMDQ_BASE.ADDR, SMMU_EVENTQ_BASE.ADDR and SMMU_PRIQ_BASE.ADDR pointers.

Cache allocation hints are present in each _BASE register and are IGNORED unless a cacheable type is used for the table or queue to which the register corresponds. The transient attribute is IMPLEMENTATION DEFINED for each _BASE register. See section 13.1.2; use of an unsupported memory type for structure or queue access might cause the access to be treated as an external abort. For example, in the case of SMMU_STRTAB_BASE, an F_STE_FETCH fault is raised.

6.3.11.1 TABLE_* attributes

The TABLE_* fields are preset when SMMU_IDR1.TABLES_PRESET == 1 and the effective attribute is unchangeable. In this case, a write of a different value to these fields is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The write of the field is ignored.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The new value is stored, making it visible to future reads of the field, but have no effect on the (fixed) access type.

Otherwise when not PRESET, the TABLE_* attributes reset to an UNKNOWN value and must be initialized by software.

These fields are Guarded by SMMU_(S_)CR0.SMMUEN and must only be changed when SMMU_(S_)CR0.SMMUEN == 0. A write to these fields when SMMU_(S_)CR0.SMMUEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The write to the fields is ignored.
- The new attribute is applied, taking effect at an UNPREDICTABLE point before the SMMU is later disabled (SMMUEN transitioned 1 to 0).

6.3.11.2 QUEUE_* attributes

The QUEUE_* fields are fixed and preset when SMMU_IDR1.QUEUES_PRESET == 1 and the effective attribute is unchangeable. In this case, a write of a different value to these fields is CONSTRAINED UNPREDICTABLE in the same way as for preset TABLE_* fields.

Otherwise when not preset, the QUEUE_* attributes reset to an UNKNOWN value and must be initialized by software.

These fields are Guarded by SMMU_(S_)CR0.EVENTQEN, SMMU_CR0.PRIQEN (for SMMU_CR1) and SMMU_(S_)CR0.CMDQEN. They must only change when access to all queues is disabled through SMMU_(S_)CR0.EVENTQEN == 0 and SMMU_CR0.PRIQEN == 0 and SMMU_(S_)CR0.CMDQEN == 0. A write to these fields when any of these queues are enabled is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The write to the fields is ignored.
- The new attribute is applied, taking effect, with respect to an enabled queue access, at an UNPREDICTABLE point before the respective queue is later disabled (the enable flag transitioned from 1 to 0).

6.3.12 SMMU_CR2

- [31:3] Reserved

- [2] PTM Private TLB Maintenance.
  - `0b0`: The SMMU participates in broadcast TLB maintenance, if implemented. See SMMU_IDR0.BTM.
  - `0b1`: The SMMU is not required to invalidate any local TLB entries on receipt of broadcast TLB maintenance operations for NS-EL1, EL2, or EL2-E2H translation regimes.
  - Broadcast invalidation for S-EL1 or EL3 translation regimes are not affected by this flag, see SMMU_S_CR2.PTM.
  - When SMMU_IDR0.BTM == 0, this field is RES0.
  - Resets to an implementation specific value. ARM recommends PTM is reset to 1 where supported, software cannot rely on this value.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [1] RECINVSID Record event C_BAD_STREAMID from invalid input StreamIDs.
  - `0b0`: C_BAD_STREAMID events are not recorded for the Non-secure programming interface.
  - `0b1`: C_BAD_STREAMID events are permitted to be recorded for the Non-secure programming interface.
  - Resets to an UNKNOWN value.

- [0] E2H Enable EL2-E2H translation regime
  - `0b0`: EL2 translation regime, without ASIDs or VMIDs.
  - `0b1`: EL2-E2H translation regime used, with ASID.
  - This field affects the STE.STRW encoding `0b10`, which selects a hypervisor translation regime for the resulting translations. The translations are tagged without ASID in EL2 mode, or with ASID in EL2-E2H mode. Note: ARM expects software to set this bit to match HCR_EL2.E2H in host PEs.
  - This bit is permitted to be cached in configuration caches and TLBs. Changes to this bit must be accompanied by invalidation of configuration and translations associated with streams configured with StreamWorld == EL2 or EL2-E2H.
  - When SMMU_IDR0.Hyp == 0, this field is RES0.
  - Resets to an UNKNOWN value.

This register is made read-only when the associated SMMU_(S_)CR0.SMMUEN is Updated to 1. This register must only be changed when SMMU_(S_)CR0.SMMUEN == 0.

A write to this register after SMMU_(S_)CR0.SMMUEN has been changed but before its Update completes is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- Apply the new value.
- Ignore the write.

When this register is changed, the new value takes effect (affects SMMU behavior corresponding to the field changed) at an UNPREDICTABLE time, bounded by a subsequent Update of SMMUEN to 1. As a side-effect of SMMUEN completing Update to 1, a prior change to this register is guaranteed to have taken effect.

6.3.12.1 PTM

A change to PTM is permitted to take effect at any time prior to a subsequent Update of SMMUEN to 1. If incoming broadcast TLB invalidates are received after PTM is changed but before Update of SMMUEN completes, it is UNPREDICTABLE whether the invalidations take effect.

Broadcast invalidations issued after SMMUEN has Updated to 1 are guaranteed to be treated with the value of PTM prior to the SMMUEN update, if PTM has not been modified after SMMUEN was written.

6.3.12.2 RECINVSID

This field only has an effect on transactions received when SMMUEN == 1, therefore a change cannot affect transactions in flight.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6.3.12.3 E2H

All EL2/EL2-E2H translation cache entries must be invalidated when E2H is changed.

Note: This can be done using the following procedure:

1. (As SMMU_CR2 is R/O when SMMU_CR0.SMMUEN == 1, ensure SMMU_CR0.SMMUEN == 0.).
2. Write E2H with new_value.
3. Issue CMD_TLBI_EL2_ALL.
4. Issue CMD_SYNC, and wait for completion.
5. (Here, no EL2/EL2-E2H translation cache entries are present.)
6. Update SMMUEN 0-1, if required.
7. Streams configured with STE.STRW == `0b10` take the new value of E2H.

Note: The behavior of the CMD_TLBI_EL2_VAA and CMD_TLBI_EL2_ASID commands depends on the value of E2H. Because, after write of SMMU_CR2, the effective action of E2H is UNPREDICTABLE until SMMUEN is transitioned to 1, it is UNPREDICTABLE whether these two commands behave according to E2H == 0 or E2H == 1. Consequently, CMD_TLBI_EL2_ALL must be used to invalidate EL2 translation cache entries.

6.3.13 SMMU_GBPA

Global ByPass Attribute

- [31] Update Update completion flag, see section 6.3.13.1.
  - This field resets to 0.
- [30:21] Reserved, RES0.
- [20] ABORT Abort all incoming transactions.
  - 0: Do not abort incoming transactions. Transactions bypass the SMMU with attributes given by other fields in this register.
  - 1: Abort all incoming transactions.
  - This field resets to an IMPLEMENTATION DEFINED value.
  - Note: An implementation can reset this field to 1, in order to implement a default deny policy on reset.
- [19:18] INSTCFG Instruction/data override.
  - `0b00`: Use incoming.
  - `0b01`: Reserved, behaves as 0b00.
  - `0b10`: Data.
  - `0b11`: Instruction.
    - INSTCFG only affects reads, writes are always output as Data.
  - This field resets to an IMPLEMENTATION DEFINED value.
- [17:16] PRIVCFG User/privileged override.
  - `0b00`: Use incoming.
  - `0b01`: Reserved, behaves as `0b00`.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b10`: Unprivileged.
- `0b11`: Privileged.
- This field resets to an IMPLEMENTATION DEFINED value.

- [15:14] Reserved, RES0.

- [13:12] SHCFG Shareability override.
  - `0b00`: Non-shareable.
  - `0b01`: Use incoming.
  - `0b10`: Outer Shareable.
  - `0b11`: Inner Shareable.
  - This field resets to an IMPLEMENTATION DEFINED value.

- [11:8] ALLOCCFG
  - `0b0xxx`: Use incoming RA/WA/TR allocation/transient hints
  - `0b1RWT`: Hints are overridden to given values:
    - Read Allocate == R
    - Write Allocate == W
    - Transient == T
  - When overridden by this field, for each of RA, WA and TR, both inner- and outer- hints are set to the same value. Because it is not architecturally possible to express hints for types that are Device or Normal Non-cacheable, this field has no effect on memory types that are not Normal-WB or Normal-WT, whether such types are provided with a transaction or overridden using MTCFG/MemAttr.
  - This field resets to an IMPLEMENTATION DEFINED value.

- [7:5] Reserved, RES0.

- [4] MTCFG Memory type override.
  - `0b0`: Use incoming memory type.
  - `0b1`: Override incoming memory type using MemAttr field.
  - This field resets to an IMPLEMENTATION DEFINED value.

- [3:0] MemAttr Memory type.
  - Encoded the same as the STE.MemAttr field.
  - This field resets to an IMPLEMENTATION DEFINED value.

This register controls the global bypass attributes used for transactions from Non-secure StreamIDs (as determined by SSD, if supported) when SMMU_CR0.SMMUEN == 0. Transactions passing through the SMMU when it is disabled might have their attributes overridden/assigned using this register.

Where Use incoming is selected, the attribute is taken from that supplied on the incoming interconnect, if supported. If the incoming interconnect does not supply the attribute, the SMMU generates a default attribute, which is selected for Use incoming. See section 13 for details.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

It is IMPLEMENTATION DEFINED whether MTCFG, SHCFG and ALLOCCFG apply to streams associated with PCIe devices or whether these attributes are treated as Use incoming for such streams regardless of the field values.

If SMMU_IDR1.ATTR_TYPES_OVR == 0, MTCFG, SHCFG, ALLOCCFG are effectively fixed as Use incoming and it is implementation specific whether these fields read as zero or a previously written value. In this case, MemAttr reads as UNKNOWN.

If SMMU_IDR1.ATTR_PERMS_OVF == 0, INSTCFG and PRIVCFG are effectively fixed as Use incoming and it is implementation specific whether these fields read as zero or a previously written value.

If the outgoing interconnect does not support a particular attribute, the value written to the corresponding field is ignored and it is implementation specific whether the field reads as zero or a previously written value.

Update resets to zero and all other fields reset to an IMPLEMENTATION DEFINED state. This allows an implementation to provide useful default transaction attributes when software leaves the SMMU uninitialized.

6.3.13.1 Update procedure

This register must be written with a single 32-bit write that simultaneously sets the Update bit to 1. Software must then poll the Update bit which is cleared when the attribute update has completed.

The Update flag allows software to determine when a change in attributes takes effect. Transactions arriving at the SMMU after completion of a GPBA update are guaranteed to take the new attributes written.

If GBPA is altered in the middle of a stream of transactions, the exact point in the sequence at which the change takes effect is UNPREDICTABLE.

Note: It is the responsibility of client devices to ensure that transactions generated prior to an update have completed, meaning that no more transactions will become globally visible to the required Shareability domain of the overridden attributes with attributes given by a previous value of the register. This is achieved in an IMPLEMENTATION DEFINED manner.

This register must only be written when Update == 0 (prior updates are complete). A write when an Update == 1, that is when a prior update is underway, is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- Update completes with any value.
    - Note: The effective attribute in use might not match that read back from this register.
- The write is ignored and update completes using the initial value.

If this register is written without simultaneously setting Update to 1, the effect is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The write is ignored.
- The written value is stored and is visible to future reads of the register, but does not affect transactions.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The written value affects transactions at an UNPREDICTABLE update point:
    - There is no guarantee that all transactions arriving at the SMMU after the write will take the new value, or that all transactions prior to the write have completed.

When this register is written (correctly observing the requirements in this section), the new value is observable to future reads of the register even if they occur before the Update has completed.

6.3.14 SMMU_AGBPA

Alternate Global ByPass Attribute

- [31:0] IMPLEMENTATION DEFINED IMPLEMENTATION DEFINED attributes to assign
    - This register allows an implementation to apply an additional non-architected attributes or tag to bypassing transactions.
    - If this field is unsupported by an implementation, it is RES0.
    - Note: ARM does not recommend that this register further modifies existing architected bypass attributes.

The process used to change contents of this register in relation to SMMU_GPBA.Update is IMPLEMENTATION DEFINED.

6.3.15 SMMU_STATUSR

Read-only.

- [31:1] Reserved, RES0.

- [0] DORMANT Dormant hint
    - `0b0`: The SMMU might have cached translation or configuration structure data, or be in the process of doing so.
    - `0b1`: The SMMU guarantees that no translation or configuration structure data is cached, and that no prefetches are in-flight.
    - Software might avoid issuing configuration invalidation or TLB invalidation commands for changes to structures made visible to the SMMU before reading this hint as 1.
    - SMMU_IDR0.DORMHINT == 0, this bit is RES0.
    - See section 3.19.1.

This register resets to zero.

6.3.16 SMMU_IRQ_CTRL

- [31:3] Reserved, RES0.

- [2] EVENTQ_IRQEN.
    - `0b0`: Interrupts from the Non-Secure Event queue are disabled.
    - `0b1`: Interrupts from the Non-secure Event queue are enabled.

- [1] PRIQ_IRQEN.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- o `0b0`: Interrupts from PRI queue are disabled.
- o `0b1`: Interrupts from PRI queue are enabled.
  Reserved if SMMU_IDR0.PRI == 0.

- [0] GERROR_IRQEN.
  - o `0b0`: Interrupts from Non-secure Global errors are disabled.
  - o `0b1`: Interrupts from Non-secure Global errors are enabled.

Resets to zero.

Each field in this register has a corresponding field in SMMU_IRQ_CTRLACK, with the same Update and observability semantics as fields in SMMU_CR0 versus SMMU_CR0ACK.

This register contains IRQ enable flags for GERROR, PRI queue and Event queue interrupt sources. These enables allow or inhibit both edge-triggered wired outputs if implemented and MSI writes if implemented.

IRQ enable flags Guard the MSI address and payload registers when MSIs supported, SMMU_IDR0.MSI == 1, which must only be changed when their respective enable flag is 0. See SMMU_GERROR_IRQ_CFG0 for details.

Completion of Update to IRQ enables guarantees the following side-effects:

- Completion of an Update of x_IRQEN from 0 to 1 guarantees that the MSI configuration in SMMU_*x*_IRQ_CFG{0,1,2} will be used for all future MSIs generated from source x. All wired or MSI interrupts that are triggered from a source relate to occurrences that happened after the completion of the Update that enabled the source. It is not permitted to trigger an interrupt that relates to an occurrence that happened before the source was enabled, even if the source was previously enabled at the time of the occurrence.
- An Update of *x*_IRQEN from 1 to 0 completes when all prior MSIs have completed. An MSI has completed when it is visible to its Shareability domain, or when it has aborted, and the abort is recorded in the appropriate SMMU_(S_)GERROR bit. Completion of this Update guarantees that no new MSI writes or wired edge events from source *x* become visible until the source is re-enabled. See section 3.18.1.

6.3.17 SMMU_IRQ_CTRLACK

Read-only.

- Same fields as SMMU_IRQ_CTRL.

Resets to zero.

Undefined bits read as zero. Fields in this register are RAZ if their corresponding SMMU_IRQ_CTRL field is Reserved.

6.3.18 SMMU_GERROR

Read-only.

- [31:9] Reserved, RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [8] SFM_ERR
  - `0b0`: The SMMU has not entered Service failure mode.
  - `0b1`: The SMMU has entered Service failure mode.
    - Traffic through the SMMU might be affected. Depending on the origin of the error, the SMMU might stop processing commands and recording events. The RAS registers describe the error.
    - Acknowledgement of this error through GERRORN does not clear this error, which is cleared in an IMPLEMENTATION DEFINED way. See 12.3.
    - SFM triggers SFM_ERR in SMMU_GERROR, and when SMMU_S_IDR1.SECURE_IMPL == 1 in SMMU_S_GERROR.

- [7] MSI_GERROR_ABT_ERR
  - `0b0`: A GERROR MSI was not terminated with an abort.
  - `0b1`: A GERROR MSI was terminated with abort.
    - Activation of this error does not affect future MSIs.
  - If SMMU_IDR0.MSI == 0, this field is RES0.

- [6] MSI_PRIQ_ABT_ERR
  - `0b0`: A PRI queue MSI was not terminated with abort.
  - `0b1`: A PRI queue MSI was terminated with abort.
    - Activation of this error does not affect future MSIs.
  - If SMMU_IDR0.PRI == 0 or SMMU_IDR0.MSI == 0, this field is RES0.

- [5] MSI_EVENTQ_ABT_ERR
  - `0b0`: An Event queue MSI was not terminated with abort.
  - `0b1`: An Event queue MSI was terminated with abort.
    - Activation of this error does not affect future MSIs.
  - If SMMU_IDR0.MSI == 0, this field is RES0.

- [4] MSI_CMDQ_ABT_ERR
  - `0b0`: A CMD_SYNC MSI was not terminated with abort.
  - `0b1`: A CMD_SYNC MSI was terminated with abort.
    - Activation of this error does not affect future MSIs.
  - If SMMU_IDR0.MSI == 0, this field is RES0.

- [3] PRIQ_ABT_ERR
  - `0b0`: An access to the PRI queue was not terminated with abort.
  - `0b1`: An access to the PRI queue was terminated with abort.
    - Page Request records might have been lost.
  - If SMMU_IDR0.PRI == 0, this field is RES0.
- [2] EVENTQ_ABT_ERR
  - `0b0`: An access to the Event queue was not terminated with abort.
  - `0b1`: An access to the Event queue was terminated with abort.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

        - Event records might have been lost.

- [1] Reserved, RES0.

- [0] CMDQ_ERR
    - `0b0`: A command that cannot be processed has not been encountered.
    - `0b1`: A command has been encountered that cannot be processed.
        - SMMU_CMDQ_CONS.ERR has been updated with a reason code and command processing has stopped.
        - Commands are not processed while this error is active.

Resets to zero.

This register, in conjunction with SMMU_(S_)GERRORN, indicates whether global error conditions exist. See section 7.5.

The SMMU toggles SMMU_(S_)GERROR[x] when an error becomes active. ARM expects software to toggle SMMU_(S_)GERRORN[x] in response, to acknowledge the error.

The SMMU does not toggle a bit when an error is already active. An error is only activated if it is in an inactive state.

Note: Software is not intended to trigger interrupts by arranging for GERRORN[x] to differ from GERROR[x]. See section 6.3.19.

6.3.19 SMMU_GERRORN

- Same fields as SMMU_GERROR

Resets to zero. Fields that are Reserved in SMMU_GERROR are also Reserved in this register.

Software must not toggle fields in this register that correspond to errors that are inactive. It is CONSTRAINED UNPREDICTABLE whether or not an SMMU activates errors if this is done.

The SMMU does not alter fields in this register. A read of this register returns the values that were last written to the defined fields of the register.

Note: Software might maintain an internal copy of the last value written to this register, for comparison against values read from SMMU_GERROR when probing for errors.

6.3.20 SMMU_GERROR_IRQ_CFG0

- [63:52] Reserved, RES0.

- [51:2] ADDR[51:2] PA of MSI target register

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The effective address has ADDR[63:48] == 0 and ADDR[1:0] == 0.
- High-order bits of the ADDR field above the system physical address size, as reported by SMMU_IDR5.OAS, are RES0.
  - Note: An implementation is not required to store these bits.

  If ADDR == 0, no MSI is sent. This allows a wired IRQ, if implemented, to be used when SMMU_(S_)IRQ_CTRL.GERROR_IRQEN == 1 instead of an MSI.

- [1:0] Reserved, RES0.

This register resets to an UNKNOWN value.

When an implementation does not support Non-secure MSIs, all fields are RES0.

SMMU_(S_)GERROR_IRQ_CFG0 is Guarded by the respective SMMU_(S_)IRQ_CTRL.GERROR_IRQEN and must only be modified when GERROR_IRQEN == 0.

A write of either while SMMU_(S_)IRQ_CTRL.GERROR_IRQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause MSIs to be written to UNPREDICTABLE addresses or with UNPREDICTABLE payload values.
- The write is ignored.

A read following such a write will return an UNKNOWN value.

These update conditions are common for all SMMU_*_IRQ_CFG{0,1,2} register sets in the SMMU with respect to their corresponding SMMU_(S_)IRQ_CTRL.*_IRQEN flag.

6.3.21 SMMU_GERROR_IRQ_CFG1

- [31:0] DATA MSI data payload.

This register resets to an UNKNOWN value.

When an implementation does not support Non-secure MSIs, all fields are RES0.

SMMU_GERROR_IRQ_CFG1 and SMMU_S_GERROR_IRQ_CFG1 are Guarded by SMMU_IRQ_CTRL.GERROR_IRQEN and SMMU_S_IRQ_CTRL.GERROR_IRQEN, respectively, and must only be modified when GERROR_IRQEN == 0.

6.3.22 SMMU_GERROR_IRQ_CFG2

- [31:6] Reserved, RES0.

- [5:4] SH[1:0] Shareability
  - `0b00`: Non-shareable.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b01`: Reserved, treated as `0b00`.
- `0b10`: Outer Shareable.
- `0b11`: Inner Sharable.
  - When MemAttr specifies a Device memory type, the contents of this field are IGNORED and the Shareability is effectively Outer Shareable.

- [3:0] MemAttr Memory type.
  - Encoded the same as the STE.MemAttr field.

This register resets to an UNKNOWN value.

MemAttr and SH allow the memory type and Shareability of the MSI to be configured, see section 3.18. The encodings of all SMMU_*_IRQ_CFG2 MemAttr and SH fields are the same. When a cacheable type is specified in MemAttr, the allocation and transient hints are IMPLEMENTATION DEFINED.

When an implementation does not support Non-secure MSIs, all fields are RES0.

SMMU_GERROR_IRQ_CFG2 and SMMU_S_GERROR_IRQ_CFG2 are Guarded by SMMU_IRQ_CTRL.GERROR_IRQEN and SMMU_S_IRQ_CTRL.GERROR_IRQEN, respectively, and must only be modified when GERROR_IRQEN == 0.

6.3.23 SMMU_STRTAB_BASE

- [63] Reserved, RES0.

- [62] RA Read Allocate hint
  - `0b0`: No Read-Allocate.
  - `0b1`: Read-Allocate.

- [61:52] Reserved, RES0.

- [51:6] ADDR[51:6] Physical address of Stream table base
  - Address bits above and below this field range are implied as zero.
  - High-order bits of the ADDR field above the system physical address size, as reported by SMMU_IDR5.OAS, are RES0.
    - Note: An implementation is not required to store these bits.
  - When a Linear Stream table is used, that is when SMMU_STRTAB_BASE_CFG.FMT == `0b00`, the effective base address is aligned by the SMMU to the table size, ignoring the least-significant bits in the ADDR range as required to do so:

    `ADDR[LOG2SIZE + 5:0] = 0.`
  - When a 2 level Stream table is used, that is when SMMU_STRTAB_BASE_CFG.FMT == `0b01`, the effective base address is aligned by the SMMU to the larger of 64 bytes or the first-level table size:

    `ADDR[MAX(5, (LOG2SIZE - SPLIT - 1 + 3)):0] = 0.`

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The alignment of ADDR is affected by the literal value of the respective SMMU_STRTAB_BASE_CFG.LOG2SIZE field and is not limited by SIDSIZE.

Note: This means that configuring a table that is larger than required by the incoming StreamID span results in some entries being unreachable, but the table is still aligned to the configured size.

- [5:0] Reserved, RES0.

When SMMU_IDR1.TABLES_PRESET == 1, this register resets to a IMPLEMENTATION DEFINED value. Otherwise this register resets to an UNKNOWN value.

When SMMU_IDR1.TABLES_PRESET == 1, this register is read-only.

Otherwise, SMMU_STRTAB_BASE is guarded by the respective SMMU_CR0.SMMUEN and must only be written when SMMU_CR0.SMMUEN == 0.

A write while SMMU_CR0.SMMUEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause STEs to be fetched from an UNPREDICTABLE address.
- The write is ignored.

A read following such a write will return an UNKNOWN value.

Access attributes of the Stream table are set using the SMMU_CR1.TABLE_* fields, a Read-Allocate hint is provided for Stream table accesses with the RA field.

6.3.24 SMMU_STRTAB_BASE_CFG

- [31:18] Reserved, RES0.

- [17:16] FMT Format of Stream table.
  - `0b00`: Linear ADDR points to an array of STEs.
  - `0b01`: 2-level ADDR points to an array of Level 1 Stream table descriptors.
  - Other values are reserved, behave as `0b00`.
  - When SMMU_IDR0.ST_LEVEL == `0b00`, this field is RES0.

- [15:11] Reserved, RES0.

- [10:6] SPLIT StreamID split point for multi-level table.
  - This field determines the split point of a 2-level Stream table, selected by the number of bits at the bottom level.
  - This field is IGNORED if FMT == `0b00`.
  - This field is RES0 when SMMU_IDR0.ST_LEVEL == `0b00`.
  - 6 4KB leaf tables.
  - 8 16KB leaf tables.
  - 10 64KB leaf tables.
  - Other values are reserved, behave as 6.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The upper-level L1STD is located using StreamID[LOG2SIZE - 1:SPLIT] and this indicates the lowest-level table which is indexed by StreamID[SPLIT - 1:0].
    - For example, selecting SPLIT == 6 causes StreamID[5:0] to be used to index the lowest level Stream table and StreamID[LOG2SIZE - 1:6] to index the upper level table.
- Note: If SPLIT ≥LOG2SIZE, a single upper-level descriptor indicates one bottom-level Stream table with $2^{LOG2SIZE}$ usable entries. The L1STD.Span value's valid range is up to SPLIT + 1, but not all of this Span is accessible, as it is not possible to use a StreamID $\geq 2^{LOG2SIZE}$.

  Note: ARM recommends that a Linear table, FMT == `0b00`, is used instead of programming SPLIT > LOG2SIZE.

- [5:0] LOG2SIZE[5:0] Table size as $\log_2$(entries)
  - The maximum StreamID value that can be used to index into the Stream table is $2^{LOG2SIZE}-1$. The StreamID range is equal to the number of STEs in a linear Stream table or the maximum sum of the STEs in all second-level tables. The number of L1STDs in the upper level of a 2-level table is $MAX(1, 2^{LOG2SIZE-SPLIT})$. Except for readback of a written value, the effective LOG2SIZE is ≤ SMMU_IDR1.SIDSIZE for the purposes of input StreamID range checking and upper/lower/linear Stream table index address calculation.

When SMMU_IDR1.TABLES_PRESET == 1 this register resets to an IMPLEMENTATION DEFINED value, otherwise it resets to an UNKNOWN value.

When SMMU_IDR1.TABLES_PRESET == 1, this register is read-only.

Otherwise, SMMU_STRTAB_BASE_CFG is Guarded by SMMU_CR0.SMMUEN and must only be modified when SMMU_CR0.SMMUEN == 0. A write while SMMU_CR0.SMMUEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause STEs to be fetched from an UNPREDICTABLE address.
- The write is ignored.

A read following such a write will return an UNKNOWN value.

A transaction having a StreamID $\geq 2^{LOG2SIZE}$ is out of range. Such a transaction is terminated with abort and a C_BAD_STREAMID event is recorded if permitted by SMMU_CR2.RECINVSID.

6.3.25 SMMU_CMDQ_BASE

- [63] Reserved, RES0.

- [62] RA Read Allocate hint.
  - `0b0`: No Read-Allocate.
  - `0b1`: Read-Allocate.

- [61:52] Reserved, RES0.

- [51:5] ADDR[51:5] Physical address of Command queue base.
  - Address bits above and below this field range are treated as zero.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- High-order bits of the ADDR field above the system physical address size, as reported by SMMU_IDR5.OAS, are RES0.
    - Note: An implementation is not required to store these bits.
- The effective base address is aligned by the SMMU to the larger of the queue size in bytes or 32 bytes, ignoring the least-significant bits of ADDR as required. ADDR bits [4:0] are treated as zero.
    - Note: For example, a queue with 2^8 entries is 4096 bytes in size so software must align an allocation, and therefore ADDR, to a 4KB boundary.

- [4:0] LOG2SIZE[4:0] Queue size as $\log_2$(entries)
    - LOG2SIZE must be less than or equal to SMMU_IDR1.CMDQS. Except for the purposes of readback of this register, any use of the value of this field is capped at the maximum, SMMU_IDR1.CMDQS.
    - The minimum size is 0, for one entry, but this must be aligned to a 32-byte (2 entry) boundary as above.

When SMMU_IDR1.QUEUES_PRESET == 1, this register resets to an IMPLEMENTATION DEFINED value. Otherwise it resets to an UNKNOWN value.

When SMMU_IDR1.QUEUES_PRESET == 1, this register is read-only.

SMMU_CMDQ_BASE is Guarded by SMMU_CR0.CMDQEN and must only be modified when SMMU_CR0.CMDQEN == 0.

A write while SMMU_CR0.CMDQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause commands to be fetched from an UNPREDICTABLE address.
- The write is ignored.

A read following such a write will return an UNKNOWN value.

Upon initialization, if SMMU_IDR1.QUEUES_PRESET == 0 then the SMMU_CMDQ_BASE.LOG2SIZE field might affect which bits of SMMU_CMDQ_CONS.RD and SMMU_CMDQ_PROD.WR can be written upon initialization. The registers must be initialized in this order:

1. Write SMMU_CMDQ_BASE to set the queue base and size
2. Write initial values to SMMU_CMDQ_CONS and SMMU_CMDQ_PROD.
3. Enable the queue with an Update of the respective SMMU_CR0.CMDQEN to 1.

This also applies to the initialization of Event queue and PRI queue registers.

Access attributes of the Command queue are set using the SMMU_CR1.QUEUE_* fields. A Read-Allocate hint is provided for Command queue accesses with the RA field.

6.3.26 SMMU_CMDQ_CONS

- [31] Reserved, RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [30:24] ERR Error reason code.
  - When a command execution error is detected, ERR is set to a reason code and then the SMMU_GERROR.CMDQ_ERR global error becomes active.
  - The value in this field is UNKNOWN when the CMDQ_ERR global error is not active.

- [23:20] Reserved, RES0.

- [QS] RD_WRAP Queue read index wrap flag.

- [QS-1:0] RD[QS-1:0] Queue read index.
  - Updated by the SMMU (consumer) indicating which command entry has just been executed.

QS == SMMU_CMDQ_BASE.LOG2SIZE and SMMU_CMDQ_BASE.LOG2SIZE ≤SMMU_IDR1.CMDQS ≤19.

This gives a configurable-sized index pointer followed immediately by the wrap bit.

If QS < 19, bits [19:QS + 1] are RAZ. When incremented by the SMMU, the RD index is always wrapped to the current queue size given by SMMU_CMDQ_BASE.LOG2SIZE.

If QS == 0 the queue has one entry. Zero bits of RD index are present and RD_WRAP is bit zero.

When SMMU_CMDQ_BASE.LOG2SIZE is increased within its valid range, the value of the bits of this register that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

This register resets to an UNKNOWN value. ARM recommends that software initializes the register to a valid value before SMMU_CR0.CMDQEN is transitioned from 0 to 1.

This register is Guarded by SMMU_CR0.CMDQEN and must only be modified when SMMU_CR0.CMDQEN == 0.

A write to this register when SMMU_CR0.CMDQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The write is ignored.
- The register takes on any value and it is UNPREDICTABLE whether it affects the SMMU Command queue internal state. The SMMU might consume commands from UNPREDICTABLE locations within the Command queue.

A read following such a write will return an UNKNOWN value.

Upon a write to this register, when SMMU_CR0.CMDQEN == 0, the ERR field is permitted to either take the written value or ignore the written value.

Note: There is no requirement for the SMMU to update this value after every command consumed. It might be updated only after an implementation specific number of commands have been consumed. However, an SMMU must ultimately update RD in finite time to indicate free space to software.

When a command execution error is detected, ERR is set to a reason code and then the respective SMMU_GERROR.CMDQ_ERR error becomes active. RD remains pointing at the infringing command for debug. The SMMU resumes processing commands after the CMDQ_ERR error is acknowledged, if the Command queue is enabled at that time. SMMU_GERROR.CMDQ_ERR has no other interaction with SMMU_CR0.CMDQEN than that a Command queue error can only be detected when the queue is enabled and therefore consuming

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

commands. A change to SMMU_CR0.CMDQEN does not affect, or acknowledge, SMMU_GERROR.CMDQ_ERR which must be explicitly acknowledged. See section 7.1.

6.3.27 SMMU_CMDQ_PROD

- [31:20] Reserved, RES0.
- [QS] WR_WRAP Command queue write index wrap flag.
- [QS-1:0] WR[QS-1:0] Command queue write index.
  - Updated by the host PE (producer) indicating the next empty space in the queue after new data.

QS == SMMU_CMDQ_BASE.LOG2SIZE, see SMMU_CMDQ_CONS.

If QS < 19, bits [19:QS + 1] are RES0. If software writes a non-zero value to these bits, the value might be stored but has no other effect. In addition, if SMMU_IDR1.CMDQS < 19, bits [19:CMDQS+1] are UNKNOWN on read.

If QS == 0 the queue has one entry. Zero bits of WR index are present and WR_WRAP is bit zero.

When software increments WR, if the index would pass off the end of the queue it must be correctly wrapped to the queue size given by QS and WR_WRAP toggled.

Note: In the degenerate case of a one-entry queue, an increment of WR consists solely of a toggle of WR_WRAP.

There is space in the queue for additional commands if:

SMMU_CMDQ_CONS.RD != SMMU_CMDQ_PROD.WR ||

SMMU_CMDQ_CONSRD_WRAP == SMMU_CMDQ_PROD.WR_WRAP

The value written to this register must only move the pointer in a manner consistent with adding N consecutive entries to the Command queue, updating WR_WRAP when appropriate.

When SMMU_CMDQ_BASE.LOG2SIZE is increased within its valid range, the value of the bits of this register that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

This register resets to an UNKNOWN value. ARM recommends that software initializes the register to a valid value before SMMU_CR0.CMDQEN is transitioned from 0 to 1.

A write to this register causes the SMMU to consider the Command queue for processing if SMMU_CR0.CMDQEN == 1 and SMMU_GERROR.CMDQ_ERR is not active.

6.3.28 SMMU_EVENTQ_BASE

- [63] Reserved, RES0.
- [62] WA Write Allocate hint.
  - `0b0`: No Write-Allocate.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b1`: Write-Allocate.

- [61:52] Reserved, RES0.

- [51:5] ADDR[51:5] PA of queue base.
  - Address bits above and below this field range are treated as zero.
  - High-order bits of the ADDR field above the system physical address size, as reported by SMMU_IDR5.OAS, are RES0.
    - Note: An implementation is not required to store these bits.
  - The effective base address is aligned by the SMMU to the larger of the queue size in bytes or 32 bytes, ignoring the least-significant bits of ADDR as required.

- [4:0] LOG2SIZE[4:0] Queue size as $\log_2$(entries)
  - LOG2SIZE is less than or equal to SMMU_IDR1.EVENTQS. Except for the purposes of readback of this register, any use of the value of this field is capped at the maximum, SMMU_IDR1.EVENTQS.

When SMMU_IDR1.QUEUES_PRESET == 1, this register resets to an IMPLEMENTATION DEFINED value. Otherwise this register resets to an UNKNOWN value.

When SMMU_IDR1.QUEUES_PRESET == 1, this register is read-only.

Otherwise, SMMU_EVENTQ_BASE is Guarded by the respective SMMU_CR0.EVENTQEN and must only be modified when EVENTQEN == 0.

A write while SMMU_CR0.EVENTQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause events to be written to UNPREDICTABLE addresses.
- The write is ignored.

A read following such a write will return an UNKNOWN value.

See SMMU_CMDQ_BASE for initialization order with respect to the PROD and CONS registers.

Events destined for an Event queue (for the appropriate Security state, if supported) are delivered into the queue if SMMU_CR0.EVENTQEN == 1 and the queue is writable. If SMMU_CR0.EVENTQEN == 0, no events are delivered into the queue. See section 7.2; some events might be lost in these situations.

Access attributes of the Event queue are set using the SMMU_CR1.QUEUE_* fields. A Write-Allocate hint is provided for Event queue accesses with the WA field.

6.3.29 SMMU_EVENTQ_CONS

- [31] OVACKFLG Overflow acknowledge flag.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Software must set this flag to the value of SMMU_EVENTQ_PROD.OVFLG when it is safe for the SMMU to report a future EVENT queue overflow. ARM recommends that this is done on initialization and after a previous Event queue overflow is handled by software.
- See section 7.4.

- [30:20] Reserved, RES0.

- [QS] RD_WRAP Event queue read index wrap flag

- [QS-1:0] RD[QS-1:0] Event queue read index.
  - Entry last read by PE, that is the first empty queue location.

QS == SMMU_EVENTQ_BASE.LOG2SIZE and SMMU_EVENTQ_BASE.LOG2SIZE ≤SMMU_IDR1.EVENTQS ≤ 19.

This gives a configurable-sized index pointer followed immediately by the wrap bit.

If QS < 19, bits [19:QS + 1] are RES0. If software writes a non-zero value to these bits, the value might be stored but has no other effect. In addition, if SMMU_IDR1.EVENTQS < 19, bits [19:EVENTQS + 1] are UNKNOWN on read.

If QS == 0 the queue has one entry. Zero bits of RD index are present and RD_WRAP is bit zero.

When software increments RD, if the index would pass off the end of the queue it must be correctly wrapped to the queue size given by QS and RD_WRAP toggled.

This register resets to an UNKNOWN value. ARM recommends that software initializes the register to a valid value before SMMU_CR0.EVENTQEN is transitioned from 0 to 1.

When SMMU_EVENTQ_BASE.LOG2SIZE is increased within its valid range, the value of the bits of this register that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

6.3.30 SMMU_EVENTQ_PROD

- [31] OVFLG Event queue overflowed
  - An Event queue overflow is indicated using this flag. This flag is toggled by the SMMU when a queue overflow is detected, if OVFLG == SMMU_EVENTQ_CONS.OVACKFLG.
  - This flag will not be updated until a prior overflow is acknowledged by setting SMMU_EVENTQ_CONS.OVACKFLG equal to OVFLG.

- [30:20] Reserved, RES0.

- [QS] WR_WRAP Event queue write index wrap flag.

- [QS-1:0] WR[QS-1:0] Event queue write index.
  - Next space to be written by SMMU

QS == SMMU_EVENTQ_BASE.LOG2SIZE, see SMMU_EVENTQ_CONS.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If QS < 19, bits [19:QS+1] are RAZ. When incremented by the SMMU, the WR index is always wrapped to the current queue size given by QS.

If QS == 0 the queue has one entry. Zero bits of WR index are present and WR_WRAP is bit zero.

When SMMU_EVENTQ_BASE.LOG2SIZE is increased within its valid range, the value of the bits of this registers that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

This register resets to an UNKNOWN value. ARM recommends that software initializes the register to a valid value before SMMU_CR0.EVENTQEN is transitioned from 0 to 1.

This register is Guarded by SMMU_CR0.EVENTQEN and must only be modified when SMMU_CR0.EVENTQEN == 0.

A write to this register when SMMU_CR0.EVENTQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The write is ignored.
- The register takes on any value and it is UNPREDICTABLE whether it affects the SMMU Event queue internal state. The SMMU might write events to UNPREDICTABLE locations within the Event queue.

A read following such a write will return an UNKNOWN value.

Note: See section 7.4 for details on queue overflow. An overflow condition is entered when a record has been discarded due to a full enabled Event queue. The following conditions do not cause an overflow condition:

- Event records discarded when the Event queue is disabled, that is when SMMU_CR0.EVENTQEN == 0.
- A stalled faulting transaction, as stall event records do not get discarded when the Event queue is full or disabled.

6.3.31 SMMU_EVENTQ_IRQ_CFG0, SMMU_EVENTQ_IRQ_CFG1, SMMU_EVENTQ_IRQ_CFG2

Similar to SMMU_GERROR_IRQ_CFG{0,1,2}, but for Event queue MSIs and Guarded by SMMU_(S_)IRQ_CTRL.EVENTQ_IRQEN. See the field details and update conditions described in SMMU_GERROR_IRQ_CFG{0,1,2} registers.

6.3.32 SMMU_PRIQ_BASE

- [63] Reserved, RES0
- [62] WA Write allocate hint.
  - `0b0`: No Write-Allocate.
  - `0b1`: Write-Allocate.
- [61:52] Reserved, RES0.
- [51:5] ADDR[51:5] PA of queue base
  - Address bits above and below this field range are implied as zero.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- High-order bits of the ADDR field above the system physical address size, as reported by SMMU_IDR5.OAS, are RES0.
    - Note: An implementation is not required to store these bits.
- The effective base address is aligned by the SMMU to the larger of the queue size in bytes or 32 bytes, ignoring the least-significant bits of ADDR as required.

- [4:0] LOG2SIZE[4:0] Queue size as $\log_2$(entries)
    - LOG2SIZE ≤SMMU_IDR1.PRIQS (which has a maximum value of 19). Except for the purposes of readback of this register, any use of this field's value is capped at SMMU_IDR1.PRIQS.

This register resets to an UNKNOWN value. SMMU_PRIQ_BASE is Guarded by SMMU_CR0.PRIQEN and must only be modified when PRIQEN == 0.

A write while SMMU_CR0.PRIQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause page requests to be written to UNPREDICTABLE addresses.
- The write is ignored.

A read following such a write will return an UNKNOWN value.

See SMMU_CMDQ_BASE for initialization order with respect to the PROD and CONS registers.

Access attributes of the PRI queue are set using the SMMU_CR1.QUEUE_* fields. A Write-Allocate hint is provided for PRI queue accesses with the WA field.

All SMMU_PRIQ_{BASE,CONS,PROD,IRQ_CFG0,IRQ_CFG1} registers are Reserved when SMMU_IDR0.PRI == 0.

6.3.33 SMMU_PRIQ_CONS

- [31] OVACKFLG Overflow acknowledge flag
    - Note: Software sets this flag to the value of SMMU_PRIQ_PROD.OVFLG when it is ready for the SMMU to report a new PRI queue overflow. ARM expects this to be done on initialization and after a previous PRI queue overflow has been handled by software.

- [30:20] Reserved, RES0.

- [QS] RD_WRAP Queue read index wrap flag

- [QS-1:0] RD[QS-1:0] Queue read index
    - Entry last read by host PE.

QS == SMMU_PRIQ_BASE.LOG2SIZE and SMMU_PRIQ_BASE.LOG2SIZE ≤SMMU_IDR1.PRIQS ≤19. This gives a configurable-sized index pointer followed immediately by the wrap bit.

If QS < 19, bits [19:QS + 1] are RES0. If software writes a non-zero value to these bits, the value might be stored but has no other effect. In addition, if SMMU_IDR1.PRIQS < 19, bits [19:PRIQS + 1] are UNKNOWN on read.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If QS == 0 the queue has one entry: zero bits of RD index are present and RD_WRAP is bit zero.

When software increments RD, if the index would pass off the end of the queue it must be correctly wrapped to the queue size given by QS and RD_WRAP toggled.

When SMMU_PRIQ_BASE.LOG2SIZE is increased within its valid range, the value of this register's bits that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

This register resets to an UNKNOWN value. ARM recommends that software initializes the register to a valid value before SMMU_CR0.PRIQEN is transitioned from 0 to 1.

6.3.34 SMMU_PRIQ_PROD

- [31] OVFLG
  - This flag is toggled by the SMMU when a queue overflow is detected, in which case one or more requests have been lost.
  - The reset value of this flag is 0.
  - An overflow condition is present when this flag is different from SMMU_PRIQ_CONS.OVACKFLG. This flag is not updated until the overflow is acknowledged by setting SMMU_PRIQ_CONS.OVACKFLG equal to OVFLG.
- [30:20] Reserved, RES0.
- [QS] WR_WRAP Queue write index wrap flag.
- [QS-1:0] WR[QS-1:0] Queue write index
  - Next space to be written by SMMU.

QS == SMMU_PRIQ_BASE.LOG2SIZE; see SMMU_PRIQ_CONS.

If QS < 19, bits [19:QS + 1] are RAZ. When incremented by the SMMU, the WR index is always wrapped to the current queue size given by QS.

If QS == 0 the queue has one entry: zero bits of WR index are present and WR_WRAP is bit zero.

When SMMU_PRIQ_BASE.LOG2SIZE is increased within its valid range, the value of the bits of this register that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

The reset value of this register is UNKNOWN. ARM recommends that software initializes the register to a valid value before transitioning SMMU_CR0.PRIQEN from 0 to 1.

SMMU_PRIQ_PROD is Guarded by SMMU_CR0.PRIQEN and must only be modified when PRIQEN == 0.

A write to this register when SMMU_CR0.PRIQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The write is ignored.
- The register takes on any value and it is UNPREDICTABLE whether it affects the SMMU Event queue internal state. The SMMU might write page requests to UNPREDICTABLE locations within the PRI queue.

A read following such a write will return an UNKNOWN value.

6.3.35 SMMU_PRIQ_IRQ_CFG0, SMMU_PRIQ_IRQ_CFG1

Similar to SMMU_GERROR_IRQ_CFG{0,1}, but for PRI queue MSIs and Guarded by SMMU_IRQ_CTRL.PRIQ_IRQEN. See the field details and update conditions described in SMMU_GERROR_IRQ_CFG{0,1}.

6.3.36 SMMU_PRIQ_IRQ_CFG2

Similar to SMMU_GERROR_IRQ_CFG2, but for PRI queue MSIs and Guarded by SMMU_(S_)IRQ_CTRL.PRIQ_IRQEN.

- [31] LO 'L Only': only interrupt if PRI message received with L bit set
  - `0b0`: Send PRI queue interrupt when PRI queue transitions from empty to non-empty.
  - `0b1`: Send PRI queue interrupt when PRI message received with L bit set.
    - When the message is written to the PRI queue, the interrupt is visible after the queue entry becomes visible. See section 3.18.
    - When the message is discarded because of a PRI queue overflow, the interrupt is generated. When the message is discarded for any other reason, the interrupt is not generated.
- [30:6] Reserved, RES0.
- [5:4] SH[1:0] Shareability.
  - `0b00`: Non-shareable.
  - `0b01`: Reserved, treated as `0b00`.
  - `0b10`: Outer Shareable.
  - `0b11`: Inner Shareable.
- [3:0] MemAttr Memory type.
  - Encoded the same as the STE.MemAttr field.

See the field details and update conditions described in SMMU_GERROR_IRQ_CFG{0,1,2}.

When an implementation does not support Non-secure MSIs, the SH and MemAttr fields are RES0.

Note: The LO field applies to wired and MSI interrupts.

6.3.37 SMMU_GATOS_CTRL

- [31:1] Reserved, RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [0] RUN Run ATOS translation.
    - Software must write this bit to 1 to initiate the translation operation, after initializing the ATOS_SID and ATOS_ADDR registers.
    - The SMMU clears the RUN flag after the translation completes and its result is visible in ATOS_PAR.
    - A write of 0 to this flag might change the value of the flag but has no other effect. Software must only write 0 to this flag when the flag is zero.

This register resets to zero.

SMMU_GATOS_{CTRL, SID, ADDR, PAR} are only present if SMMU_IDR0.ATOS == 1; otherwise, they are Reserved. See section 9 for more information on the overall behavior of ATOS operations.

RUN is Guarded by SMMU_CR0.SMMUEN and must only be set when SMMUEN == 1 and RUN == 0.

ATOS_CTRL.RUN must not be set to 1 after SMMU_S_CR0.SMMUEN has been written to 0, including at a point prior to the completion of an update to SMMUEN. Behavior of doing so is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The write is ignored.
- The value is stored and visible for readback, but no other effect occurs and RUN is not cleared by the SMMU unless SMMUEN is later Updated to 1.

Between software writing ATOS_CTRL.RUN == 1 and observing the SMMU having cleared it to 0 again:

- Writes to ATOS_SID, ATOS_ADDR and VATOS_SEL (if appropriate) are CONSTRAINED UNPREDICTABLE and one of the following occurs:
    - The write is ignored.
    - Translation is performed with any value of the written register. After completion, ATOS_PAR is UNKNOWN.

      Note: HTTU might have been performed to an UNPREDICTABLE set of TTDs that could otherwise have been updated using any given value of the written register.
- Writes to ATOS_CTRL are CONSTRAINED UNPREDICTABLE and one of the following occurs:
    - The write is ignored.
    - The value is stored and visible for readback, but translation is unaffected and completes normally. Note: If RUN == 0 was written, it is not possible to determine that the translation has completed.
- Reads of ATOS_PAR return UNKNOWN.

The completion of an Update of SMMUEN from 1 to 0 while RUN == 1 is contingent on the completion of the ATOS operation, which clears RUN. Clearing SMMUEN while RUN == 1 is CONSTRAINED UNPREDICTABLE and the completion of the SMMUEN == 0 update ensures one of the following behaviors:

- The operation has completed normally and a valid translation/fault response is reported.
- The ATOS operation has been aborted, reporting ATOS_PAR.FAULT == 1 and ATOS_PAR.FAULTCODE == INTERNAL_ERR.
    - The translation table walks for the ongoing ATOS translation might have been partially performed. If HTTU was performed during the translation, an UNPREDICTABLE set of the TTDs relevant to the translation table walks might have been updated.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

An Update of SMMUEN from 0 to 1 clears RUN, if RUN was set while SMMUEN == 0. Completion of the Update occurs after RUN has been cleared for all relevant ATOS register groups.

6.3.38 SMMU_GATOS_SID

- [63:53] Reserved, RES0.

- [52] SSID_VALID SubstreamID valid.
  - If SMMU_IDR1.SSIDSIZE == 0, this field is RES0.

- [51:32] SUBSTREAMID SubstreamID of request.
  - If SMMU_IDR1.SSIDSIZE < 20, bits [51:32 + SMMU_IDR1.SSIDSIZE] are RES0.

- [31:0] STREAMID StreamID of request.
  - This is written with the StreamID (used to locate translations/CDs) of the request later submitted to SMMU_GATOS_ADDR.
  - If SMMU_IDR1.SID_SIZE < 32, bits [31:SMMU_IDR1.SID_SIZE] are RES0.

This register resets to an UNKNOWN value.

Bits of SubstreamID and StreamID outside of the supported range are RES0.

This register is Guarded by SMMU_GATOS_CTRL.RUN and must only be altered when RUN == 0.

6.3.39 SMMU_GATOS_ADDR

- [63:12] ADDR Requested input address
  .
- [11:10] TYPE Request type.
  - `0b00` Reserved.
  - `0b01` Stage 1 (VA to IPA).
  - `0b10` Stage 2 (IPA to PA).
  - `0b11` Stage 1 and stage 2 (VA to PA).
  - Use of a Reserved value results in an INV_REQ ATOS error.

- [9] PnU Privileged or User access.
  - `0b0`: Unprivileged.
  - `0b1`: Privileged.

- [8] RnW Read/write access.
  - `0b0`: Write.
  - `0b1`: Read.

- [7] InD Instruction/data access.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b0`: Data.
- `0b1`: Instruction.
- This bit is IGNORED if RnW == 0, and the effective InD for writes is Data.

- [6] HTTUI Inhibit hardware TTD flag update.
  - `0b0`: Flag update (HTTU) might occur, where supported by the SMMU, according to HA:HD configuration fields at stage 1 and stage 2.
  - `0b1`: HTTU is inhibited, regardless of HA/HD configuration.
    - The ATOS operation causes no state change and is passive.

- [5:0] Reserved, RES0.

This register resets to an UNKNOWN value.

This register is Guarded by SMMU_GATOS_CTRL.RUN and must only be altered when RUN == 0.

The PnU and InD attributes are not affected by the STE.INSTCFG or STE.PRIVCFG overrides.

6.3.40 SMMU_GATOS_PAR

Read-only.

This result register encodes both successful results and error results, the format is determined by the FAULT field:

- [0] FAULT Fault/error status
  - 0b0: No fault.
  - `0b1`: Fault or translation error.

When FAULT == 0, a successful result is present:

- [63:56] ATTR[7:0] Memory attributes, in MAIR format

- [55:52] Reserved, RES0.

- [51:12] ADDR[51:12] Result address.
  - Address bits above and below [51:12] are treated as zero.
  - In SMMUv3.0, bits [51:48] are RES0.

- [11] Size Translation page/block size.

- [10] Reserved, RES0.
  - Note: This bit is the NS field in SMMU_S_GATOS_PAR.

- [9:8] SH Shareability attribute.
  - `0b00`: Non-shareable.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b01`: Reserved.
- `0b10`: Outer Shareable.
- `0b11`: Inner Shareable.
  - Note: Shareability is returned as Outer Shareable when ATTR selects any Device type.

- [7:1] Reserved, RES0.

The Size field allows the page, block, or section size of the translation to be determined, using an encoding in the ADDR field. The Size flag indicates that:

- `0b0`: Translation is 4KB
- `0b1`: Translation is determined by position of lowest 1 bit in ADDR field

The translated address is aligned to the translation size (appropriate number of LSBs zeroed) and then, if the size is greater than 4KB, a single bit is set such that its position, N, denotes the translation size, where $2^{(N+1)}$ == size in bytes.

Note: For example, if Size == 1 and ADDR[14:12] == 0 and ADDR[15] == 1, the lowest set bit is 15 so the translation size is 2^{15+1}, or 64KB. In this case, the actual output address must be aligned to 64KB by masking out bit 15. Similarly, an output address with ADDR[13:12] == `0b10` denotes a page of size 2^{13+1}, or 16KB, and the output address is taken from ADDR[14] upwards.

An implementation that does not support all of the defined attributes is permitted to return the behavior that the cache supports, instead of the exact value from the translation table entries. Similarly, an implementation might return the translation page/block size that is cached rather than the size that is determined from the translation table entries.

The memory attributes and Shareability that are returned in ATTR and SH are determined from the translation tables without including STE overrides that might be configured for the given stream:

- When ATOS_ADDR.TYPE == stage 1, the stage 1 translation table attributes are returned.
- When ATOS_ADDR.TYPE == stage 2, the stage 2 translation table attributes are returned. In this case, the allocation and transient hints in ATTR are:
  - RA == WA == 1
  - TR == 0
  - Note: The stage 2 TTD.MemAttr[3:0] field does not encode RA/WA/TR.
- When ATOS_ADDR.TYPE == stage 1 and stage 2, the attributes returned are those from stage 1 combined with stage 2 (where combined is as defined in section 13).

When FAULT == 1, the translation has failed and a fault syndrome is present:

- [63:60] IMPDEF IMPLEMENTATION DEFINED fault data.

- [59:52] Reserved, RES0.

- [51:12] FADDR[51:12] Stage 2 fault page address.
  - The value returned in FADDR depends upon the cause of the fault. See section 9.1.4 for details.
  - In SMMUv3.0, bits [51:48] are RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [11:4] FAULTCODE Fault/error code.
  - See section 9.1.4 for details.

- [3] Reserved, RES0.

- [2:1] REASON Class of activity causing fault
  - This indicates the stage and reason for the fault. See section 9.1.4 for details.
  - `0b00`: Stage 1 translation-related fault occurred, or miscellaneous non-translation fault not attributable to a translation stage (for example, F_BAD_STE).
  - `0b01`: CD: Stage 2 fault occurred because of a CD fetch
  - `0b10`: TT: Stage 2 fault occurred because of a stage 1 translation table walk
  - `0b11`: IN: Stage 2 fault occurred because of the input address to stage 2 (output address from successful stage 1 translation table walk, or address given in ATOS_ADDR for stage 2-only translation)

The content of ATOS_PAR registers is UNKNOWN if values in the ATOS register group are modified after a translation has been initiated by setting ATOS_CTRL.RUN == 1. See section 9.1.4.

This register resets to an UNKNOWN value.

This register has an UNKNOWN value if read when ATOS_CTRL.RUN == 1.

6.3.41 SMMU_VATOS_SEL

- [31:16] Reserved, RES0.

- [15:0] VMID VMID associated with the VM that is using the VATOS interface
  - When SMMU_IDR0.VMID16 == 0, bits [15:8] of this field are RES0.

When requests are made through VATOS, the VMID field of the STE selected in the request is compared against this field. The request is denied if the values do not match (indicating a lookup for a StreamID not assigned to the VM granted the VATOS interface).

This register resets to an UNKNOWN value. This register is Guarded by SMMU_VATOS_CTRL.RUN and must only be altered when RUN == 0.

This register is RES0 when SMMU_IDR0.VATOS == 0.

6.3.42 SMMU_S_IDR0

Read-only.

- [31:26] Reserved, RES0.

- [25:24] STALL_MODEL Stalling fault model support.
  - Encoded identically to SMMU_IDR0.STALL_MODEL, this field indicates the SMMU support for the Stall model and the Terminate model.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- For more information, see SMMU_S_CR0.NSSTALLD.

- [23:14] Reserved, RES0.

- [13] MSI Secure Message Signalled Interrupts are supported
  - `0b0`: The SMMU supports wired interrupt notifications only for Secure events and GERROR.
    - The MSI fields in SMMU_S_EVENTQ_IRQ_CFGn and SMMU_S_GERROR_IRQ_CFGn are RES0.
  - `0b1`: Message Signalled Interrupts are supported for Secure events and GERROR.
    - Note: ARM strongly recommends that an implementation supports Non-secure MSIs (SMMU_IDR0.MSI == 1) if Secure MSIs are supported.

- [12:0] Reserved, RES0.

6.3.43 SMMU_S_IDR1

Read-only.

- [31] SECURE_IMPL Security implemented.
  - `0b0`: The SMMU implements a single Security state.
    - All SMMU_S_* secure registers are RAZ/WI.
  - `0b1`: The SMMU implements two Security states.
    - When SECURE_IMPL == 1, stage 1 must be supported, that is SMMU_IDR0.S1P == 1.

- [30:6] Reserved, RES0.
  - Note: If the SMMU implementation supports SubstreamIDs, the size of the SubstreamID that is provided for Secure StreamIDs is the same as the size that is provided for Non-secure StreamIDs. Therefore, there is no separate Secure SSIDSIZE option.

- [5:0] S_SIDSIZE Max bits of Secure StreamID.
  - Equivalent to SMMU_IDR1.SIDSIZE and encoded the same way, this field determines the maximum Secure StreamID value and therefore the maximum size of the Secure Stream table.

6.3.44 SMMU_S_IDR2

Read-only.

- [31:0] Reserved, RES0.

6.3.45 SMMU_S_IDR3

Read-only.

- [31:7] Reserved, RES0.
- [6] SAMS Secure ATS Maintenance Support
  - `0b0`: If SMMU_IDR0.ATS == 1, the CMD_ATC_INV command is supported when issued on the Non-secure and Secure Command queues. If SMMU_IDR0.PRI == 1, the CMD_PRI_RESP command is supported, when issued on the Non-secure and Secure Command queues.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- o `0b1`: If SMMU_IDR0.ATS == 1, the CMD_ATC_INV command is supported when issued on the Non-secure Command queue, but raises CERROR_ILL when issued on the Secure Command queue. If SMMU_IDR0.PRI == 1, the CMD_PRI_RESP command is supported when issued on the Non-secure Command queue, but raises CERROR_ILL when issued on the Secure Command queue.

- [5:0] Reserved, RES0.

6.3.46 SMMU_S_IDR4

Read-only.

- [31:0] IMPLEMENTATION DEFINED.

The contents of this register are IMPLEMENTATION DEFINED and can be used to identify the presence of other IMPLEMENTATION DEFINED register regions elsewhere in the memory map.

6.3.47 SMMU_S_CR0

- [31:10] Reserved, RES0.

- [9] NSSTALLD Non-secure stall disable
  - o `0b0`: Non-secure programming interface might use Stall model.
  - o `0b1`: Non-secure programming interface prohibited from using Stall model.
    - When SMMU_S_IDR0.STALL_MODEL == `0b00`, setting this bit modifies the Non-secure behavior so that only the Terminate model is available for Non-secure streams and SMMU_IDR0.STALL_MODEL reads as `0b01`. Otherwise, if NSSTALLD == 0, SMMU_IDR0.STALL_MODEL == SMMU_S_IDR0.STALL_MODEL.
  - o When SMMU_S_IDR0.STALL_MODEL != `0b00`, this bit is RES0 and SMMU_IDR0.STALL_MODEL == SMMU_S_IDR0.STALL_MODEL.
    - Note: A reserved SMMU_S_CR0 bit is not reflected into SMMU_S_CR0ACK.

- [8:6] Reserved, RES0.

- [5] SIF Secure Instruction Fetch.
  - o `0b0`: Secure transactions might exit the SMMU as a Non-secure instruction fetch.
  - o `0b1`: Secure transactions determined to be Non-secure instruction fetch are treated as a Permission fault.
    This field causes transactions from a Secure stream that are determined to be an instruction fetch, after INSTCFG fields are applied, to experience a Permission fault if their effective output NS attribute is Non-secure (NS == 1, after NSCFG fields are applied).
    - When translation is disabled because SMMUEN == 0, the transaction is terminated with abort and no F_PERMISSION is recorded.
    - When SMMUEN is set, one of the following occurs and, if the Event queue is writable, an F_PERMISSION event is recorded:
      - o If stream translation is disabled (STE.Config selects bypass, including the case where Config[2:0] == `0b101` and STE.S1DSS causes stage 1 to be skipped,

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

behaving as though Config[2:0] == `0b100`), the faulting transaction is terminated with abort.

- If stream translation is applied (STE.Config enables stage 1 and STE.S1DSS does not cause stage 1 translation to be skipped), CD.{A,R,S} govern stall and terminate behavior of the transaction.

Note: The fault event is a stage 1 permission fault as, by definition, a Secure transaction has only stage 1 configuration.
This bit is permitted to be cached in a TLB or configuration cache and an Update of this bit requires invalidation of all Secure TLB entries and configuration caches.

- [4] Reserved, RES0.
- [3] CMDQEN Enable Secure Command queue processing.
  - `0b0`: Processing of commands from the Non-secure Command queue is disabled.
  - `0b1`: Processing of commands from the Non-secure Command queue is enabled.
- [2] EVENTQEN Enable Secure Event queue writes.
  - `0b0`: Writes to the Non-secure Event queue are disabled.
  - `0b1`: Writes to the Non-secure Event queue are enabled.
- [1] Reserved, RES0.
- [0] SMMUEN Secure SMMU enable.
  - `0b0`: All Secure streams bypasses the SMMU, with attributes determined from SMMU_GBPA.
  - `0b1`: All Secure streams are checked against configuration structures, and might undergo translation.

Resets to zero.

The Update procedure, with respect to flags reflected into SMMU_S_CR0ACK, is the same as for SMMU_CR0.

The update side-effects of CMDQEN, EVENTQEN and SMMUEN fields are similar to their respective equivalents in SMMU_CR0.

6.3.47.1 NSSTALLD

This bit has no Update side-effects. NSSTALLD prevents Non-secure configuration from using the Stall model, therefore stall-related commands are unavailable and use of STE and CD stall configuration renders the structures ILLEGAL.

This bit is permitted to be cached in configuration caches.

This bit must not be modified when any of the following could occur:

- Non-secure CMD_STALL_TERM or CMD_RESUME commands have been submitted to the Non-secure Command queue, therefore might be being processed.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Non-secure transactions are being translated through structures configured to stall faults.

A change to this bit takes effect at an UNPREDICTABLE point prior to Update completion. If changed while the above stall-related activities are occurring, it is UNPREDICTABLE whether transactions and commands behave in manner corresponding to either value of this bit, until Update of this bit completes.

Update completes when it is guaranteed that a new transaction or command will behave in a manner relating to the new value of this bit.

Note: Secure software is expected to Update this bit before Non-secure software can access the SMMU.

6.3.47.2 SIF

This bit has no Update side-effects.

For a transaction that bypasses translation, a change to SIF is guaranteed to be visible with respect to that transaction after the new value of SIF is acknowledged in SMMU_S_CR0ACK.SIF.

For a transaction that undergoes translation, a change to SIF is guaranteed to be visible with respect to that transaction only after the completion of a TLB invalidation of a scope related to that transaction, where the TLB invalidation is made visible to the SMMU after the new value of SIF is acknowledged in SMMU_S_CR0ACK.SIF.

6.3.48 SMMU_S_CR0ACK

Read-only.

- Same fields as SMMU_S_CR0.

Resets to zero.

Undefined bits read as zero. Fields in this register are RAZ if their corresponding SMMU_S_CR0 field is IGNORED.

An Update to a field in SMMU_S_CR0 is considered complete, along with any side-effects, when the respective field in this register is observed to take the new value.

6.3.49 SMMU_S_CR1

- [31:12] Reserved, RES0.

- [11:10] TABLE_SH[1:0] Secure Stream table access Shareability.
  - `0b00`: Non-shareable.
  - `0b01`: Reserved, treated as `0b00`.
  - `0b10`: Outer Shareable.
  - `0b11`: Inner Shareable
    - Note: When SMMU_S_CR1.TABLE_OC == `0b00` and SMMU_S_CR1.TABLE_IC == `0b00`, this field is IGNORED and behaves as Outer Shareable.

- [9:8] TABLE_OC[1:0] Secure Stream table access Outer Cacheability.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b00`: Non-cacheable.
- `0b01`: Write-Back Cacheable.
- `0b10`: Write-Through Cacheable.
- `0b11`: Reserved, treated as `0b00`.

- [7:6] TABLE_IC[1:0] Secure Stream table access Inner Cacheability.
  - `0b00`: Non-cacheable.
  - `0b01`: Write-Back Cacheable.
  - `0b10`: Write-Through Cacheable.
  - `0b11`: Reserved, treated as `0b00`.

- [5:4] QUEUE_SH[1:0] Secure queue access Shareability.
  - `0b00`: Non-shareable.
  - `0b01`: Reserved, treated as `0b00`.
  - `0b10`: Outer Shareable.
  - `0b11`: Inner Shareable.
    - When SMMU_S_CR1.QUEUE_OC == `0b00` and SMMU_S_CR1.QUEUE_IC == `0b00`, this field is IGNORED and behaves as Outer Shareability.

- [3:2] QUEUE_OC[1:0] Secure queue access Outer Cacheability.
  - `0b00`: Non-cacheable.
  - `0b01`: Write-Back Cacheable.
  - `0b10`: Write-Through Cacheable.
  - `0b11`: Reserved, treated as `0b00`.

- [1:0] QUEUE_IC[1:0] Secure queue access Inner Cacheability.
  - `0b00`: Non-cacheable
  - `0b01`: Write-Back Cacheable
  - `0b10`: Write-Through Cacheable
  - `0b11`: Reserved, treated as `0b00`.

The TABLE_* fields set the attributes for access to memory through the SMMU_S_STRTAB_BASE.ADDR pointer. The QUEUE_* fields set the attributes for access to memory through SMMU_S_CMDQ_BASE.ADDR and SMMU_S_EVENTQ_BASE.ADDR pointers.

Cache allocation hints are present in each BASE register and are IGNORED unless a cacheable type is used for the table or queue to which the register corresponds. The transient attribute is IMPLEMENTATION DEFINED for each _BASE register. See section 13.1.2. Use of an unsupported memory type for structure or queue access might cause the access to be treated as an external abort. For example, in the case of SMMU_S_STRTAB_BASE, a F_STE_FETCH fault is raised.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6.3.50 SMMU_S_CR2

- [31:3] Reserved, RES0.

- [2] PTM Private TLB Maintenance
  - `0b0`: The SMMU participates in broadcast TLB maintenance, if implemented.
  - `0b1`: The SMMU is not required to invalidate any local TLB entries on receipt of broadcast TLB maintenance operations for S-EL1 or EL3 translation regimes.
  - Broadcast invalidation for NS-EL1, EL2 or EL2-E2H translation regimes are not affected by this flag, see SMMU_CR2.PTM.
  - When SMMU_IDR0.BTM == 0, this field is RES0.
  - Resets to an implementation specific value. ARM recommends PTM is reset to 1, software cannot rely on this value.

- [1] RECINVSID Record event C_BAD_STREAMID from invalid input StreamIDs
  - `0b0`: C_BAD_STREAMID events are not recorded for the Secure programming interface.
  - `0b1`: C_BAD_STREAMID events are permitted to be recorded for the Secure programming interface.
  - Resets to an UNKNOWN value.

- [0] Reserved, RES0.

PTM and RECINVSID are Guarded by SMMU_S_CR0.SMMUEN and must only be changed when SMMU_S_CR0.SMMUEN == 0. A write when SMMU_S_CR0.SMMUEN == 1 has the same behavior as stated in SMMU_CR2.

6.3.51 SMMU_S_GBPA

Secure Global Bypass Attributes.

- [31] Update Update completion flag, see section 6.3.13.1.
  - This field resets to 0.

- [30:21] Reserved, RES0.

- [20] ABORT Abort all incoming transactions.
  - `0b0`: Do not abort incoming transactions. Transactions bypass the SMMU with attributes given by other fields in this register.
  - `0b1`: Abort all incoming transactions.
  - This field resets to an IMPLEMENTATION DEFINED value.
  - Note: An implementation can reset this field to 1, in order to implement a default deny policy on reset.

- [19:18] INSTCFG Instruction/data override.
  - `0b00`: Use incoming.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b01`: Reserved, behaves as `0b00`.
- `0b10`: Data.
- `0b11`: Instruction.
  - INSTCFG only affects reads. Writes are always output as Data.
- This field resets to an IMPLEMENTATION DEFINED value.

- [17:16] PRIVCFG User/privileged override.
  - `0b00`: Use incoming.
  - `0b01`: Reserved, behaves as `0b00`.
  - `0b10`: Unprivileged.
  - `0b11`: Privileged.
  - This field resets to an IMPLEMENTATION DEFINED value.

- [15:14] NSCFG NS override
  - `0b00`: Use incoming.
  - `0b01`: Reserved, behaves as `0b00`.
  - `0b10`: Secure.
  - `0b11`: Non-secure.
  - This field resets to an IMPLEMENTATION DEFINED value.

- [13:12] SHCFG Shareability override.
  - `0b00`: Non-shareable.
  - `0b01`: Use incoming.
  - `0b10`: Outer Shareable.
  - `0b11`: Inner Shareable.
  - This field resets to an IMPLEMENTATION DEFINED value.

- [11:8] ALLOCCFG
  - `0b0xxx`: Use incoming RA, WA, TR allocation and transient hints
  - `0b1RWT`: Hints are overridden to given values:
    - Read Allocate == R
    - Write Allocate == W
    - Transient == T
  - When overridden by this field, for each of RA, WA, and TR, both inner- and outer- hints are set to the same value. Because it is not architecturally possible to express hints for types that are Device or Normal Non- cacheable, this field has no effect on memory types that are not Normal-WB or Normal-WT, whether such types are provided with a transaction or overridden using MTCFG/MemAttr.
  - This field resets to an IMPLEMENTATION DEFINED value.

- [7:5] Reserved, RES0.

- [4] MTCFG Memory type override.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b0`: Use incoming memory type.
- `0b1`: Override incoming memory type using MemAttr field.
- This field resets to an IMPLEMENTATION DEFINED value.

- [3:0] MemAttr Memory type.
  - Encoded the same as the STE.MemAttr field.
  - This field resets to an IMPLEMENTATION DEFINED value.

This register controls the Secure global bypass attributes used for transactions from Secure StreamIDs when SMMU_S_CR0.SMMUEN == 0. Transactions passing through the SMMU when it is disabled might have their attributes overridden or assigned using this register.

If SMMU_IDR1.ATTR_PERMS_OVR == 0, NSCFG is fixed as “Use incoming” and it is implementation specific whether this field reads as zero or a previously written value.

6.3.52 SMMU_S_AGBPA

Secure Alternate Global ByPass Attribute

- [31:0] IMPLEMENTATION DEFINED IMPLEMENTATION DEFINED attributes to assign
  - This register allows an implementation to apply an additional non-architected attributes or tag to bypassing transactions.
  - If this field is unsupported by an implementation, it is RES0.
  - Note: ARM does not recommend that this register further modifies existing architected bypass attributes.

The process used to change contents of this register in relation to SMMU_S_GPBA.Update is IMPLEMENTATION DEFINED

6.3.53 SMMU_S_INIT

- [31:1] Reserved, RES0.

- [0] INV_ALL Invalidate all cache and TLB contents.
  - For writes:
    - `0b0`: If INV_ALL == 0, ignored.
    - `0b1`: SMMU-global invalidation is performed for all configuration and translation caches for all translation regimes and Security states.
  - For reads:
    - `0b0`: There is no outstanding global invalidation operation.
    - `0b1`: There is an outstanding global invalidation operation.
  - Note: This field can be used to simplify Secure software that otherwise makes no use of the SMMU but must safely initialize the SMMU for use by Non-secure software. See section 3.11.
  - Note: When SMMU_S_IDR1.SECURE_IMPL == 1, but no Secure software exists, ARM strongly recommends this register is exposed for use by Non-secure initialization software.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

This register resets to zero.

A write of 1 to INV_ALL when SMMU_CR0.SMMUEN == 1 or SMMU_S_CR0.SMMUEN == 1, or an update of either SMMUEN to 1 is in progress, is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The write is ignored.
- The invalidation operation occurs and completes, with INV_ALL reset to 0 on completion.

An update of SMMUEN to 1 while an INV_ALL invalidation operation is underway has a CONSTRAINED UNPREDICTABLE effect on the invalidation operation and has one of the following behaviors:

- The invalidation operation completes successfully, with INV_ALL reset to 0 after completion.
- The invalidation operation might not affect any cache or TLB entries, and INV_ALL is reset to 0 by the SMMU.

After INV_ALL is written to 1, a write of 0 before the invalidation operation is observed to have completed is CONSTRAINED UNPREDICTABLE has and one of the following behaviors:

- The invalidation operation does not take place.
- The invalidation operation completes successfully but it cannot be determined when this completion occurs.

When observing the conditions in this section, a write of INV_ALL to 1 causes an invalidation of all cache and TLB entries that are present before the write and on completion of the invalidation the SMMU resets INV_ALL to 0, including when an invalidation operation was already underway before the write.

Note: If the conditions regarding SMMUEN are observed correctly, a write of 1 to INV_ALL is guaranteed to invalidate the SMMU caches and reset INV_ALL to 0 when complete.

The completion of an INV_ALL invalidation is not required to depend on the completion of any outstanding transactions.

An INV_ALL invalidation operation affects locked configuration and translation cache entries, if an implementation supports locking of cache entries.

6.3.54 SMMU_S_IRQ_CTRL

- [31:3] Reserved, RES0.
- [2] EVENTQ_IRQEN Secure Event queue interrupt enable
  - `0b0`: Interrupts from the Secure Event queue are disabled.
  - `0b1`: Interrupts from the Secure Event queue are enabled.
- [1] Reserved, RES0.
- [0] GERROR_IRQEN Secure GERROR interrupt enable
  - `0b0`: Interrupts from Secure Global errors are disabled.
  - `0b1`: Interrupts from Secure Global errors are enabled.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

This register is similar to SMMU_IRQ_CTRL, but controls interrupts from the Secure programming interface. It relates to SMMU_S_IRQ_CTRLACK in the same way as SMMU_IRQ_CTRL relates to SMMU_IRQ_CTRLACK.

6.3.55 SMMU_S_IRQ_CTRLACK

Read-only.

- Same fields as SMMU_S_IRQ_CTRL.

Resets to zero.

Undefined bits read as zero. Fields in this register are RAZ if the corresponding SMMU_S_IRQ_CTRL field is Reserved.

6.3.56 SMMU_S_GERROR

Read-only.

- [31:9] Reserved, RES0.

- [8] SFM_ERR
  - `0b0`: The SMMU has not entered Service fail mode.
  - `0b1`: The SMMU has entered Service failure mode.
    - Traffic through the SMMU might be affected. Depending on the origin of the error, the SMMU might stop processing commands and recording events. The RAS registers describe the error.
    - Acknowledgement of this error through GERRORN does not clear this error, which is cleared in an IMPLEMENTATION DEFINED way. See section 12.3.
  - SFM triggers SFM_ERR in SMMU_GERROR, and when SMMU_S_IDR1.SECURE_IMPL == 1 in SMMU_S_GERROR.

- [7] MSI_GERROR_ABT_ERR
  - `0b0`: A Secure GERROR MSI was not terminated with an abort.
  - `0b1`: A Secure GERROR MSI was terminated with abort.
    - Activation of this error does not affect future MSIs.
  - If SMMU_S_IDR0.MSI == 0, this field is RES0.

- [6] Reserved, RES0.

- [5] MSI_EVENTQ_ABT_ERR
  - `0b0`: A Secure EVENT queue MSI was not terminated with abort.
  - `0b1`: A Secure EVENT queue MSI was terminated with abort.
    - Activation of this error does not affect future MSIs.
  - If SMMU_S_IDR0.MSI == 0, this field is RES0.

- [4] MSI_CMDQ_ABT_ERR
  - `0b0`: A Secure CMD_SYNC MSI was not terminated with abort.
  - `0b1`: A Secure CMD_SYNC MSI was terminated with abort.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

        - Activation of this error does not affect future MSIs.
    - If SMMU_S_IDR0.MSI == 0, this field is RES0.

- [3] Reserved, RES0.

- [2] EVENTQ_ABT_ERR
    - `0b0`: An access to the Secure Event queue was not terminated with abort.
    - `0b1`: An access to the Secure Event queue was terminated with abort.
    - Event records might have been lost

- [1] Reserved, RES0.

- [0] CMDQ_ERR
    - `0b0`: A command that cannot be processed has not been encountered on the Secure Command queue.
    - `0b1`: A command has been encountered that cannot be processed on the Secure Command queue.
        - SMMU_S_CMDQ_CONS.ERR has been updated with a reason code and command processing has stopped.
    - Commands are not processed while this error is active.

Resets to zero.

6.3.57 SMMU_S_GERRORN

- Same fields as SMMU_S_GERROR

Resets to zero. Fields that are RES0 in SMMU_S_GERROR are also RES0 in this register.

Software must not toggle fields in this register that correspond to errors that are inactive. It is CONSTRAINED UNPREDICTABLE whether or not an SMMU activates errors if this is done.

6.3.58 SMMU_S_GERROR_IRQ_CFG0, SMMU_S_GERROR_IRQ_CFG1, SMMU_S_GERROR_IRQ_CFG2

Similar to SMMU_GERROR_IRQ_{0,1,2}, for Secure global error reporting when Secure MSIs are supported.

6.3.59 SMMU_S_STRTAB_BASE

- [63] Reserved, RES0.

- [62] RA Read allocate hint
    - `0b0`: No Read-Allocate.
    - `0b1`: Read-Allocate.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [61:52] Reserved, RES0.

- [51:6] ADDR[51:6] Physical address of Secure Stream table base
  - Address bits above and below this field range are treated as zero.
  - High-order bits of the ADDR field above the system physical address size, as reported by SMMU_IDR5.OAS, are RES0.
    - Note: An implementation is not required to store these bits.
  - When a Linear Stream table is used, that is when SMMU_S_STRTAB_BASE_CFG.FMT == `0b00`, the effective base address is aligned by the SMMU to the table size, ignoring the least-significant bits in the ADDR range as required to do so:

    `ADDR[LOG2SIZE + 5:0] = 0`.
  - When a 2-level Stream table is used, that is when SMMU_S_STRTAB_BASE_CFG.FMT == `0b01`, the effective base address is aligned by the SMMU to the larger of 64 bytes or the first-level table size:

    `ADDR[MAX(5, (LOG2SIZE - SPLIT - 1 + 3)):0] = 0`.

    The alignment of ADDR is affected by the literal value of the respective SMMU_S_STRTAB_BASE_CFG.LOG2SIZE field and is not limited by S_SIDSIZE.

    Note: This means that configuring a table that is larger than required by the incoming StreamID span results in some entries being unreachable, but the table is still aligned to the configured size.

- [5:0] Reserved, RES0.

When SMMU_IDR1.TABLES_PRESET == 1, this register resets to a IMPLEMENTATION DEFINED value. Otherwise this register resets to an UNKNOWN value.

When SMMU_IDR1.TABLES_PRESET == 1, this register is read-only.

Otherwise, SMMU_S_STRTAB_BASE is Guarded by the respective SMMU_S_CR0.SMMUEN and must only be written when SMMU_S_CR0.SMMUEN == 0.

A write while SMMU_S_CR0.SMMUEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause STEs to be fetched from an UNPREDICTABLE address.
- The write is ignored.

A read following such a write will return an UNKNOWN value.

Access attributes of the Stream table are set using the SMMU_S_CR1.TABLE_* fields, a Read-Allocate hint is provided for Stream table accesses with the RA field.

6.3.60 SMMU_S_STRTAB_BASE_CFG

- [31:18] Reserved, RES0.

- [17:16] FMT Format of Secure Stream table.
  - `0b00`: Linear ADDR points to an array of STEs

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b01`: 2-level ADDR points to an array of Level 1 Stream table descriptors
- Other values are reserved, behave as `0b00`.
- When SMMU_IDR0.ST_LEVEL == `0b00`, this field is RES0.

- [15:11] Reserved, RES0.

- [10:6] SPLIT StreamID split point for multi-level table.
  - This field determines the split point of a 2-level Stream table, selected by the number of bits at the bottom level.
  - This field is IGNORED if FMT == `0b00`.
  - This field is RES0 when SMMU_IDR0.ST_LEVEL == `0b00`.
  - 6 4KB leaf tables.
  - 8 16KB leaf tables.
  - 10 64KB leaf tables.
  - Other values are reserved, behave as 6.
  - The upper-level L1STD is located using StreamID[LOG2SIZE-1:SPLIT] and this indicates the lowest-level table which is indexed by StreamID[SPLIT-1:0].
    - For example, selecting SPLIT == 6 causes StreamID[5:0] to be used to index the lowest level Stream table and StreamID[LOG2SIZE-1:6] to index the upper level table.
  - Note: If SPLIT ≥LOG2SIZE, a single upper-level descriptor indicates one bottom-level Secure Stream table with $2^{LOG2SIZE}$ usable entries. The valid range of the L1STD.Span value is up to SPLIT + 1, but not all of this Span is accessible, because it is not possible to use a StreamID $\geq 2^{LOG2SIZE}$.

    Note: ARM recommends that a Linear table, FMT == `0b00`, is used instead of programming SPLIT > LOG2SIZE.

- [5:0] LOG2SIZE[5:0] Table size as $\log_2$(entries)
  - The maximum Secure StreamID value that can be used to index into the Secure Stream table is $2^{LOG2SIZE} - 1$. The Secure StreamID range is equal to the number of STEs in a linear Secure Stream table or the maximum sum of the STEs in all second-level tables. The number of L1STDs in the upper level of a 2-level table is $\mathrm{MAX}(1, 2^{LOG2SIZE-SPLIT})$. Except for readback of a written value, the effective LOG2SIZE is ≤SMMU_S_IDR1.S_SIDSIZE for the purposes of input StreamID range checking and upper/lower/linear Stream table index address calculation.

When SMMU_IDR1.TABLES_PRESET == 1 this register resets to an IMPLEMENTATION DEFINED value, otherwise it resets to an UNKNOWN value.

When SMMU_IDR1.TABLES_PRESET == 1, this register is read-only.

Otherwise, SMMU_S_STRTAB_BASE_CFG is Guarded by SMMU_S_CR0.SMMUEN and must only be modified when SMMU_S_CR0.SMMUEN == 0. A write while SMMU_S_CR0.SMMUEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause STEs to be fetched from an UNPREDICTABLE address.
- The write is ignored.

A read following such a write will return an UNKNOWN value.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

A transaction having a StreamID ≥$2^{LOG2SIZE}$ is out of range. Such a transaction is terminated with an abort, and a C_BAD_STREAMID event is recorded if permitted by SMMU_S_CR2.RECINVSID.

6.3.61 SMMU_S_CMDQ_BASE

- [63] Reserved, RES0.

- [62] RA Read Allocate hint.
  - `0b0`: No Read-Allocate.
  - `0b1`: Read-Allocate.

- [61:52] Reserved, RES0.

- [51:5] ADDR[51:5] Physical address of Secure Command queue base.
  - Address bits above and below this field range are implied as zero.
  - High-order bits of the ADDR field above the system physical address size, as reported by SMMU_IDR5.OAS, are RES0.
    - Note: An implementation is not required to store these bits.
  - The effective base address is aligned by the SMMU to the larger of the queue size in bytes or 32 bytes, ignoring the least-significant bits of ADDR as required.
    - Note: For example, a queue with 2^8 entries is 4096 bytes in size so software must align an allocation, and therefore ADDR, to a 4KB boundary.

- [4:0] LOG2SIZE[4:0] Queue size as $\log_2$(entries)
  - LOG2SIZE must be less than or equal to SMMU_IDR1.CMDQS. Except for the purposes of readback of this register, any use of this field's value is capped at the maximum, SMMU_IDR1.CMDQS.
  - The minimum size is 0, for one entry, but this must be aligned to a 32-byte (2 entry) boundary as above.

When SMMU_IDR1.QUEUES_PRESET == 1, this register resets to an IMPLEMENTATION DEFINED value. Otherwise it resets to an UNKNOWN value.

When SMMU_IDR1.QUEUES_PRESET == 1, this register is read-only.

SMMU_S_CMDQ_BASE is Guarded by SMMU_S_CR0.CMDQEN and must only be modified when SMMU_S_CR0.CMDQEN == 0.

A write while SMMU_S_CR0.CMDQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause commands to be fetched from an UNPREDICTABLE address.
- The write is ignored.

A read following such a write will return an UNKNOWN value.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Upon initialization, if SMMU_IDR1.QUEUES_PRESET == 0 then the SMMU_S_CMDQ_BASE.LOG2SIZE field might affect which bits of SMMU_S_CMDQ_CONS.RD and SMMU_S_CMDQ_PROD.WR can be written upon initialization. The registers must be initialized in this order:

4. Write SMMU_S_CMDQ_BASE to set the queue base and size
5. Write initial values to SMMU_S_CMDQ_CONS and SMMU_S_CMDQ_PROD.
6. Enable the queue with an Update of the respective SMMU_S_CR0.CMDQEN to 1.

This also applies to the initialization of Secure Event queue registers.

Access attributes of the Secure Command queue are set using the SMMU_S_CR1.QUEUE_* fields. A Read-Allocate hint is provided for Secure Command queue accesses with the RA field.

6.3.62 SMMU_S_CMDQ_CONS

- [31] Reserved, RES0.
- [30:24] ERR Error reason code.
  - When a command execution error is detected, ERR is set to a reason code and then the SMMU_S_GERROR.CMDQ_ERR global error becomes active.
  - The value in this field is UNKNOWN when the CMDQ_ERR global error is not active.
- [23:20] Reserved, RES0.
- [QS] RD_WRAP Secure Command queue read index wrap flag.
- [QS-1:0] RD[QS-1:0] Secure Command queue read index.
  - Updated by the SMMU (consumer) indicating which command entry has just been executed.

QS == SMMU_S_CMDQ_BASE.LOG2SIZE and SMMU_S_CMDQ_BASE.LOG2SIZE ≤SMMU_IDR1.CMDQS ≤ 19.

This gives a configurable-sized index pointer followed immediately by the wrap bit.

If QS < 19, bits [19:QS + 1] are RAZ. When incremented by the SMMU, the RD index is always wrapped to the current queue size given by SMMU_S_CMDQ_BASE.LOG2SIZE.

If QS == 0 the queue has one entry: zero bits of RD index are present and RD_WRAP is bit zero.

When SMMU_S_CMDQ_BASE.LOG2SIZE is increased within its valid range, the value of this register's bits that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

This register resets to an UNKNOWN value. ARM recommends that software initializes the register to a valid value before SMMU_S_CR0.CMDQEN is transitioned from 0 to 1.

This register is Guarded by SMMU_S_CR0.CMDQEN and must only be modified when SMMU_S_CR0.CMDQEN == 0.

A write to this register when SMMU_S_CR0.CMDQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The write is ignored.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The register takes on any value and it is UNPREDICTABLE whether it affects the internal state of the SMMU Secure Command queue. The SMMU might consume commands from UNPREDICTABLE locations within the Secure Command queue.

A read following such a write will return an UNKNOWN value.

Upon a write to this register, when SMMU_S_CR0.CMDQEN == 0, the ERR field is permitted to either take the written value or ignore the written value.

Note: There is no requirement for the SMMU to update this value after every command consumed, it might be updated only after an implementation specific number of commands have been consumed. However, an SMMU must ultimately update RD in finite time to indicate free space to software.

When a command execution error is detected, ERR is set to a reason code and then the respective SMMU_S_GERROR.CMDQ_ERR error becomes active. RD remains pointing at the infringing command for debug. The SMMU resumes processing commands after the CMDQ_ERR error is acknowledged, if the Secure Command queue is enabled at that time. SMMU_S_GERROR.CMDQ_ERR has no other interaction with SMMU_S_CR0.CMDQEN than that a Secure Command queue error can only be detected when the queue is enabled and therefore consuming commands. A change to CMDQEN does not affect, or acknowledge, SMMU_S_GERROR.CMDQ_ERR which must be explicitly acknowledged. See section 7.1.

6.3.63 SMMU_S_CMDQ_PROD

- [31:20] Reserved, RES0.
- [QS] WR_WRAP Secure Command queue write index wrap flag.
- [QS-1:0] WR[QS-1:0] Secure Command queue write index.
  - Updated by the host PE (producer) indicating the next empty space in the queue after the new data.

QS == SMMU_S_CMDQ_BASE.LOG2SIZE, see SMMU_S_CMDQ_CONS.

If QS < 19, bits [19:QS + 1] are RES0. If software writes a non-zero value to these bits, the value might be stored but has no other effect. In addition, if SMMU_IDR1.CMDQS < 19, bits [19:CMDQS + 1] are UNKNOWN on read.

If QS == 0 the queue has one entry: zero bits of WR index are present and WR_WRAP is bit zero.

When software increments WR, if the index would pass off the end of the queue it must be correctly wrapped to the queue size given by QS and WR_WRAP toggled.

Note: In the limit case of a one-entry queue, an increment of WR consists solely of a toggle of WR_WRAP.

There is space in the queue for additional commands if:

SMMU_S_CMDQ_CONS.RD != SMMU_S_CMDQ_PROD.WR ||

SMMU_S_CMDQ_CONS.RD_WRAP == SMMU_S_CMDQ_PROD.WR_WRAP

The value written to this register must only move the pointer in a manner consistent with adding N consecutive entries to the command queue, updating WR_WRAP when appropriate.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When SMMU_S_CMDQ_BASE.LOG2SIZE is increased within its valid range, the value of the bits of this register that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

This register resets to an UNKNOWN value. ARM recommends that software initializes the register to a valid value before SMMU_S_CR0.CMDQEN is transitioned from 0 to 1.

A write to this register causes the SMMU to consider the Command queue for processing if SMMU_S_CR0.CMDQEN == 1 and SMMU_S_GERROR.CMDQ_ERR is not active.

6.3.64 SMMU_S_EVENTQ_BASE

- [63] Reserved, RES0.
- [62] WA Write Allocate hint.
  - `0b0`: No Write-Allocate.
  - `0b1`: Write-Allocate.
- [61:52] Reserved, RES0.
- [51:5] ADDR[51:5] Physical address of Secure Event queue base.
  - Address bits above and below this field range are treated as zero.
  - High-order bits of the ADDR field above the system physical address size, as reported by SMMU_IDR5.OAS, are RES0.
    - Note: An implementation is not required to store these bits.
  - The effective base address is aligned by the SMMU to the larger of the queue size in bytes or 32 bytes, ignoring the least-significant bits of ADDR as required.
- [4:0] LOG2SIZE[4:0] Queue size as $\log_2$(entries)
  - LOG2SIZE is less than or equal to SMMU_IDR1.EVENTQS. Except for the purposes of readback of this register, any use of the value of this field is capped at the maximum, SMMU_IDR1.EVENTQS.

When SMMU_IDR1.QUEUES_PRESET == 1, this register resets to an IMPLEMENTATION DEFINED value. Otherwise this register resets to an UNKNOWN value.

When SMMU_IDR1.QUEUES_PRESET == 1, this register is read-only.

Otherwise, SMMU_S_EVENTQ_BASE is Guarded by the respective SMMU_S_CR0.EVENTQEN and must only be modified when EVENTQEN == 0.

A write while SMMU_S_CR0.EVENTQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The register takes on any value, which might cause events to be written to UNPREDICTABLE addresses.
- The write is ignored.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

A read following such a write will return an UNKNOWN value.

See SMMU_S_CMDQ_BASE for initialization order with respect to the PROD and CONS registers.

Events destined for an Event queue (for the appropriate Security state, if supported) are delivered into the queue if SMMU_S_CR0.EVENTQEN == 1 and the queue is writable. If SMMU_S_CR0.EVENTQEN == 0, no events are delivered into the queue. See section 7.2, some events might be lost in these situations.

Access attributes of the Secure Event queue are set using the SMMU_S_CR1.QUEUE_* fields. A Write-Allocate hint is provided for Secure Event queue accesses with the WA field.

6.3.65 SMMU_S_EVENTQ_CONS

- [31] OVACKFLG Overflow acknowledge flag.
  - Software must set this flag to the value of SMMU_S_EVENTQ_PROD.OVFLG when it is safe for the SMMU to report a future Event queue overflow. ARM recommends that this is be done on initialization and after a previous Event queue overflow is handled by software.
  - See section 7.4.
- [30:20] Reserved, RES0.
- [QS] RD_WRAP Secure Event queue read index wrap flag
- [QS-1:0] RD[QS-1:0] Secure Event queue read index.
  - Entry last read by PE, that is the first empty queue location.

QS == SMMU_S_EVENTQ_BASE.LOG2SIZE and SMMU_S_EVENTQ_BASE.LOG2SIZE ≤ SMMU_IDR1.EVENTQS ≤19.

This gives a configurable-sized index pointer followed immediately by the wrap bit.

If QS < 19, bits [19:QS + 1] are RES0. If software writes a non-zero value to these bits, the value might be stored but has no other effect. In addition, if SMMU_IDR1.EVENTQS < 19, bits [19:EVENTQS + 1] are UNKNOWN on read.

If QS == 0 the queue has one entry: zero bits of RD index are present and RD_WRAP is bit zero.

When software increments RD, if the index would pass off the end of the queue it must be correctly wrapped to the queue size given by QS and RD_WRAP toggled.

This register resets to an UNKNOWN value. ARM recommends that software initializes the register to a valid value before SMMU_S_CR0.EVENTQEN is transitioned from 0 to 1.

When SMMU_S_EVENTQ_BASE.LOG2SIZE is increased within its valid range, the value of the bits of this register that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6.3.66 SMMU_S_EVENTQ_PROD

- [31] OVFLG Secure Event queue overflowed
  - An Secure Event queue overflow is indicated using this flag. This flag is toggled by the SMMU when a queue overflow is detected, if OVFLG == SMMU_EVENTQ_CONS.OVACKFLG.
  - This flag will not be updated until a prior overflow is acknowledged by setting SMMU_EVENTQ_CONS.OVACKFLG equal to OVFLG.

- [30:20] Reserved, RES0.

- [QS] WR_WRAP Secure Event queue write index wrap flag.

- [QS-1:0] WR[QS-1:0] Secure Event queue write index.
  - This bit indicates the next space to be written by SMMU.

QS == SMMU_S_EVENTQ_BASE.LOG2SIZE, see SMMU_S_EVENTQ_CONS.

If QS < 19, bits [19:QS + 1] are RAZ. When incremented by the SMMU, the WR index is always wrapped to the current queue size given by QS.

If QS == 0 the queue has one entry: zero bits of WR index are present and WR_WRAP is bit zero.

When SMMU_S_EVENTQ_BASE.LOG2SIZE is increased within its valid range, the value of the bits of this register that were previously above the old wrap flag position are UNKNOWN and when it is decreased, the value of the bits from the wrap flag downward are the effective truncation of the value in the old field.

This register resets to an UNKNOWN value. ARM recommends that software initializes the register to a valid value before SMMU_S_CR0.EVENTQEN is transitioned from 0 to 1.

This register is Guarded by SMMU_S_CR0.EVENTQEN and must only be modified when SMMU_S_CR0.EVENTQEN == 0.

A write to this register when SMMU_S_CR0.EVENTQEN == 1 is CONSTRAINED UNPREDICTABLE and has one of the following behaviors:

- The write is ignored.
- The register takes on any value and it is UNPREDICTABLE whether it affects the internal state of the SMMU Secure Event queue. The SMMU might write events to UNPREDICTABLE locations within the Secure Event queue.

A read following such a write will return an UNKNOWN value.

Note: See section 7.4 for details on queue overflow. An overflow condition is entered when a record has been discarded due to a full enabled Secure Event queue. The following conditions do not cause an overflow condition:

- Event records discarded when the Secure Event queue is disabled, that is when SMMU_S_CR0.EVENTQEN == 0.
- A stalled faulting transaction, as stall event records do not get discarded when the queue is full or disabled.

6.3.67 SMMU_S_EVENTQ_IRQ_CFG0, SMMU_S_EVENTQ_IRQ_CFG1, SMMU_S_EVENTQ_IRQ_CFG2

Similar to SMMU_EVENTQ_IRQ_CFG{0,1,2}, for Secure Event queue interrupts when Secure MSIs are supported.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6.3.68 SMMU_S_GATOS_CTRL

- [31:1] Reserved, RES0.

- [0] RUN Run ATOS translation.
  - ARM recommends that software writes this bit to 1 to initiate the translation operation, after initializing the ATOS_SID and ATOS_ADDR registers.
  - The SMMU clears the RUN flag after the translation completes and its result is visible in ATOS_PAR.
  - A write of 0 to this flag might change the value of the flag but has no other effect. Software must only write 0 to this flag when the flag is zero.

This register resets to zero.

SMMU_S_GATOS_{CTRL, SID, ADDR, PAR} are only present if SMMU_IDR0.ATOS == 1, otherwise they are Reserved. See section 9 for more information on the overall behavior of ATOS operations.

RUN is Guarded by SMMU_S_CR0.SMMUEN and must only be set when SMMUEN == 1 and RUN == 0.

ATOS_CTRL.RUN must not be set to 1 after SMMU_S_CR0.SMMUEN has been written to 0, including at a point prior to the completion of an update to SMMUEN. Behavior of doing so is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The write is ignored.
- The value is stored and visible for readback, but no other effect occurs and RUN is not cleared by the SMMU unless SMMUEN is later Updated to 1.

Between software writing ATOS_CTRL.RUN == 1 and observing the SMMU having cleared it to 0 again,

- Writes to ATOS_SID, ATOS_ADDR and VATOS_SEL (if appropriate) are CONSTRAINED UNPREDICTABLE and one of the following occurs:
  - The write is ignored.
  - Translation is performed with any value of the written register. After completion, ATOS_PAR is UNKNOWN.

    Note: HTTU might have been performed to an UNPREDICTABLE set of TTDs that could otherwise have been updated using any given value of the written register.
- Writes to ATOS_CTRL are CONSTRAINED UNPREDICTABLE and one of the following occurs:
  - The write is ignored.
  - The value is stored and visible for readback, but translation is unaffected and completes normally. Note: If RUN == 0 was written, it is not possible to determine that the translation has completed.
- Reads of ATOS_PAR return an UNKNOWN value.

The completion of an Update of SMMUEN from 1 to 0 while RUN == 1 is contingent on the completion of the ATOS operation, which clears RUN. Clearing SMMUEN while RUN == 1 is CONSTRAINED UNPREDICTABLE and the completion of the SMMUEN == 0 update ensures one of the following behaviors:

- The operation has completed normally and a valid translation or fault response is reported.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The ATOS operation has been aborted, reporting ATOS_PAR.FAULT == 1 and ATOS_PAR.FAULTCODE == INTERNAL_ERR.
    - The translation table walks for the ongoing ATOS translation might have been partially performed. If HTTU was performed during the translation, an UNPREDICTABLE set of the TTDs relevant to the translation table walks might have been updated.

An Update of SMMUEN from 0 to 1 clears RUN, if RUN was set while SMMUEN == 0 == 0. Completion of the Update occurs after RUN has been cleared for all relevant ATOS register groups.

6.3.69 SMMU_S_GATOS_SID

- [63:53] Reserved, RES0.

- [53] SSEC Secure stream lookup
    - `0b0`: Non-secure stream lookup: STREAMID field is a Non-secure StreamID.
    - `0b1`: Secure stream lookup: STREAMID field is a Secure StreamID.

- [52] SSID_VALID SubstreamID valid.
    - If SMMU_IDR1.SSIDSIZE == 0, this field is RES0.

- [51:32] SUBSTREAMID SubstreamID of request.
    - If SMMU_IDR1.SSIDSIZE < 20, bits [51:32 + SMMU_IDR1.SSIDSIZE] are RES0.

- [31:0] STREAMID StreamID of request.
    - This is written with the StreamID (used to locate translations/CDs) of the request later submitted to SMMU_S_GATOS_ADDR.
    - If SMMU_IDR1.SID_SIZE < 32 and SMMU_S_IDR1.S_SIDSIZE < 32, bits [31:`MAX(SMMU_IDR1.SID_SIZE, SMMU_S_IDR1.S_SID_SIZE)`] are RES0.

This register resets to an UNKNOWN value.

Bits of SUBSTREAMID and STREAMID outside of the supported range are RES0.

This register is Guarded by SMMU_S_GATOS_CTRL.RUN and must only be altered when RUN == 0.

6.3.70 SMMU_S_GATOS_ADDR

- [63:12] ADDR Requested input address
  .
- [11:10] TYPE Request type.
    - `0b00` Reserved.
    - `0b01` Stage 1 (VA to IPA).
    - `0b10` Stage 2 (IPA to PA).
    - `0b11` Stage 1 and stage 2 (VA to PA).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Use of a Reserved value results in an INV_REQ ATOS error.

- [9] PnU Privileged or User access.
  - `0b0`: Unprivileged.
  - `0b1`: Privileged.

- [8] RnW Read/Write access.
  - `0b0`: Write.
  - `0b1`: Read.

- [7] InD Instruction/Data access.
  - `0b0`: Data.
  - `0b1`: Instruction.
  - This bit is IGNORED if RnW == 0, and the effective InD for writes is Data.

- [6] HTTUI Inhibit hardware TTD flag update.
  - `0b0`: Flag update (HTTU) might occur, where supported by the SMMU, according to the HA and HD configuration fields at stage 1 and stage 2.
  - `0b1`: HTTU is inhibited, regardless of HA and HD configuration.
    - The ATOS operation causes no state change.

- [5:0] Reserved, RES0.

This register resets to an UNKNOWN value.

This register is Guarded by SMMU_S_GATOS_CTRL.RUN and must only be altered when RUN == 0.

6.3.71 SMMU_S_GATOS_PAR

Read-only.

This result register encodes both successful results and error results, the format is determined by the FAULT field:

- [0] FAULT Fault/error status
  - `0b0`: No fault.
  - `0b1`: Fault/translation error.

When FAULT == 0, a successful result is present:

- [63:56] ATTR[7:0] Memory attributes, in MAIR format

- [55:52] Reserved, RES0.

- [51:12] ADDR[51:12] Result address.
  - Address bits above and below [51:12] are treated as zero.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- In SMMUv3.0, bits [51:48] are RES0.

- [11] Size Translation page/block size.

- [10] NS NS attribute determined from translation table.
  - Note: This bit is RES0 in SMM_GATOS_PAR and SMMU_VATOS_PAR.

- [9:8] SH Shareability attribute.
  - `0b00`: Non-shareable.
  - `0b01`: Reserved.
  - `0b10`: Outer Shareable.
  - `0b11`: Inner Shareable.
    - Note: Shareability is returned as Outer Shareable when ATTR selects any Device type.

- [7:1] Reserved, RES0.

The Size field allows the page, block, or section size of the translation to be determined, using an encoding in the ADDR field. The Size flag indicates that:

- 0: Translation is 4KB
- 1: Translation is determined by position of lowest 1 bit in ADDR field

The translated address is aligned to the translation size (appropriate number of LSBs zeroed) and then, if the size is greater than 4KB, a single bit is set such that its position, N, denotes the translation size, where $2^{(N+1)}$ == size in bytes.

Note: For example, if Size == 1 and ADDR[14:12] == 0 and ADDR[15] == 1, the lowest set bit is 15 so the translation size is 2^{15+1}, or 64KB. In this case, ARM expects software to align the actual output address to 64KB by masking out bit 15. Similarly, an output address with ADDR[13:12] == `0b10` denotes a page of size 2^{13+1}, or 16KB, and the output address is taken from ADDR[47] upwards.

An implementation that does not support all of the defined attributes is permitted to return the behavior that the cache supports, instead of the exact value from the translation table entries. Similarly, an implementation might return the translation page or block size that is cached rather than the size that is determined from the translation table entries.

The returned memory attributes and Shareability are determined from the translation tables without including STE overrides that might be configured for the given stream.

- When ATOS_ADDR.TYPE == stage 1, the stage 1 translation table attributes are returned.
- When ATOS_ADDR.TYPE == stage 2, the stage 2 translation table attributes are returned. In this case, the allocation and transient hints in ATTR are:
  - RA == WA == 1
  - TR == 0
  - Note: The stage 2 TTD.MemAttr[3:0] field does not encode RA/WA/TR.
- When ATOS_ADDR.TYPE == stage 1 and stage 2, the attributes returned are those from stage 1 combined with stage 2 (where combined is as defined in Section 13).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When FAULT == 1, the translation has failed and a fault syndrome is present:

- [63:60] IMPDEF IMPLEMENTATION DEFINED fault data
- [59:52] Reserved, RES0.
- [51:12] FADDR[51:12] Stage 2 fault page address.
  - The value returned in FADDR depends upon the cause of the fault. See section 9.1.4 for details.
  - In SMMUv3.0, bits [51:48] are RES0.
- [11:4] FAULTCODE Fault/error code.
  - See section 9.1.4 for details.
- [3] Reserved, RES0.
- [2:1] REASON Class of activity causing fault
  - This indicates the stage and reason for the fault. See section 9.1.4 for details.
  - `0b00`: Stage 1 translation-related fault occurred, or miscellaneous non-translation fault not attributable to a translation stage (for example F_BAD_STE).
  - `0b01`: CD: Stage 2 fault occurred because of a CD fetch
  - `0b10`: TT: Stage 2 fault occurred because of a stage 1 translation table walk
  - `0b11`: IN: Stage 2 fault occurred because of the input address to stage 2 (output address from successful stage 1 translation table walk, or address given in ATOS_ADDR for stage 2-only translation)

The content of ATOS_PAR registers is UNKNOWN if values in the ATOS register group are modified after a translation has been initiated by setting ATOS_CTRL.RUN == 1. See section 9.1.4.

This register resets to an UNKNOWN value.

This register has an UNKNOWN value if read when ATOS_CTRL.RUN == 1.

6.3.72 ID_REGS

Register offsets `0xFD0-0xFFC` are defined as a read-only identification register space. For ARM implementations of the SMMU architecture the assignment of this register space, and naming of registers in this space, is consistent with the ARM identification scheme for ARM CoreLink and ARM CoreSight components. ARM strongly recommends that other implementers also use this scheme to provide a consistent software discovery model.

For ARM implementations, the following assignment of fields, consistent with CoreSight ID registers [8], is used:

Offset	*Name*	*Field*	*Value*	*Meaning*
`0xFF0`	SMMU_CIDR0, Component ID0	[7:0]	`0x0D`	Preamble
`0xFF4`	SMMU_CIDR1, Component ID1	[7:4]	`0xF`	CLASS

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Offset	Name	Field	Value	Meaning
		[3:0]	0x0	Preamble
0xFF8	SMMU_CIDR2, Component ID2	[7:0]	0x05	Preamble
0xFFC	SMMU_CIDR3, Component ID3	[7:0]	0xB1	Preamble
0xFE0	SMMU_PIDR0, Peripheral ID0	[7:0]	IMPDEF	PART_0: bits [7:0] of the Part number
0xFE4	SMMU_PIDR1, Peripheral ID1	[7:4]	IMPDEF	DES_0: bits [3:0] of the JEP106 Designer code
		[3:0]	IMPDEF	PART_1: bits [11:8] of the Part number
0xFE8	SMMU_PIDR2, Peripheral ID2	[7:4]	IMPDEF	REVISION
		[3]	1	JEDEC-assigned value for DES always used
		[2:0]	IMPDEF	DES_1: bits [6:4] bits of the JEP106 Designer code
0xFEC	SMMU_PIDR3, Peripheral ID3	[7:4]	IMPDEF	REVAND
		[3:0]	IMPDEF	CMOD
0xFD0	SMMU_PIDR4, Peripheral ID4	[7:4]	0	SIZE
		[3:0]	IMPDEF	DES_2: JEP106 Designer continuation code
0xFD4	SMMU_PIDR5, Peripheral ID5		RES0	Reserved
0xFD8	SMMU_PIDR6, Peripheral ID6		RES0	Reserved
0xFDC	SMMU_PIDR7, Peripheral ID7		RES0	Reserved

Fields outside of those defined in this table are RES0.

Note: The Designer code fields (DES_*) fields for ARM-designed implementations use continuation code 0x4 and Designer code 0x3B.

Note: Non-ARM implementations that follow this CoreSight ID register layout must set the Designer fields appropriate to the implementer.

6.3.73 SMMU_VATOS_CTRL, SMMU_VATOS_SID, SMMU_VATOS_ADDR, SMMU_VATOS_PAR

SMMU_VATOS_CTRL, SMMU_VATOS_SID, SMMU_VATOS_ADDR and SMMU_VATOS_PAR are present only if SMMU_IDR0.VATOS == 1.

See section 9 for more information on the behavior of ATOS operations.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

7 Event queue, Faults and Errors

There are three ways that unexpected or erroneous events can be reported to software:

1. Commands given to the SMMU might be in some way incorrect and the Command queue has a mechanism to report such problems.
2. A set of configuration errors and faults are recorded in the Event queue. These include events that arise from incoming device traffic, such as a configuration error (discovered when configuration is fetched on receipt of device traffic) or a page fault arising from the device traffic address.
3. A global register-based SMMU_GERROR mechanism reports events arising from a failure to record into the Event or PRI queues and other catastrophic events that cannot be written to memory. This might happen when the Event queue base pointer incorrectly indicates non-existent memory, or queue overflow.

7.1 Command queue errors

The Command queue entry formats are described in chapter 4, which defines what is considered to be a valid command. Commands are consumed in-order and when a command error or command fetch abort is detected:

- Command consumption stops.
- Older commands (prior to the erroneous command in the queue) fully complete their execution and can be observed to have been Consumed.

  Note: By definition, earlier commands must have been Consumed for a later invalid command to be indicated using CMDQ_CONS.
- SMMU_(S_)CMDQ_CONS.RD remains pointing to the erroneous command in the Command queue. The CONS index is not permitted to increment in the case where a command fetch experiences an external abort, meaning that external aborts on command read are synchronous.
- The SMMU_(S_)CMDQ_CONS.ERR field is updated to provide an error code.
- The global Command queue Error is triggered: SMMU_(S_)GERROR.CMDQ_ERR is activated (indicating that SMMU_(S_)CMDQ_CONS.ERR has been updated). Commands are not consumed from the affected queue while this error is active, see 7.5 and 6.3.18.
- Commands newer than the erroneous command have no effect, and if they have been fetched they are discarded.

ARM recommends that software rectifies the cause of the command error, then restarts command processing by acknowledging the CMDQ_ERR by writing an appropriate value to SMMU_(S_)GERRORN. Software is not required to write SMMU_(S_)CMDQ_PROD to re-trigger command processing.

While the error is active, additional commands might be submitted and CMDQ_PROD might be moved backwards as far as the CMDQ_CONS position so that previously-submitted but non-consumed commands are removed from the queue. This is the only condition in which it is permissible to change CMDQ_PROD in a manner that is not an increment/wrap while the queue is enabled, see section 3.21.2.

Commands at or after the CMDQ_CONS.RD position are fetched or re-fetched after command processing is restarted by acknowledging CMDQ_ERR.

Note: A Command queue error is completely recoverable and, when the erroneous command is fixed or replaced with a valid command, consumption can be restarted. Older commands are unaffected by a later Command queue error. It is acceptable for software to change the contents of the erroneous command and newer commands while the error is active, as such commands are re-fetched when command processing is restarted.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The SMMU_(S_)CMDQ_CONS.ERR field is updated with the error reason code after detecting a command error. The error reason code is made visible to software before the SMMU makes the global error visible.

Note: It is not possible for software to observe that an error has occurred through GERROR without being able to observe the error code.

Note: If software polls SMMU_(S_)CMDQ_CONS waiting for prior commands to complete, ARM recommends that GERROR.CMDQ_ERR is also be checked to avoid an infinite loop in the event of a command error.

A Command queue error can be raised for the following reasons:

SMMU_(S_)CMDQ_CONS. ERR value	Error name	Cause
`0x00`	CERROR_NONE	No error This value is defined for completeness only, and is not provided by the SMMU in any error case.
`0x01`	CERROR_ILL	Command illegal and cannot be correctly consumed because of an: • Unknown command opcode, including architecture-defined commands irrelevant to an implementation without a certain feature or Security state. For example, stage 1 invalidation commands on an SMMU without stage 1, secure invalidation commands from the Non-secure command queue. • Command valid but reserved field or invalid value used.
`0x02`	CERROR_ABT	Abort on command fetch: an external abort was returned for a read of the command queue, if an interconnect can detect and report such an event.
`0x03`	CERROR_ATC_INV_SYNC	A CMD_SYNC failed to successfully complete one or more previous CMD_ATC_INV commands. This is the result of an ATS Invalidation Completion timeout. See section 3.9.1.2.

7.2 Event queue recorded faults and events

Three categories of events might be recorded into the Event queue:

- Configuration errors.
- Faults from the translation process.
- Miscellaneous.

Configuration errors result from improper register, STE or CD contents and are associated with the translation of an incoming transaction. An improper configuration in memory is not reported until the data structures are used in response to a transaction. A fault from translation is reported only when a transaction attempts a translation. Any

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

incoming transaction might cause **at most one** event report which might be a configuration error or, if configuration is all valid, one of several kinds of translation fault (which might be applicable to stage 1 or stage 2).

An SMMU implementation might prefetch configuration and TLB entries in an implementation specific manner. A prefetched erroneous configuration does not record a configuration error (even if prefetched in response to an explicit CMD_PREFETCH_* command). A translation fault or configuration error is recorded only on receipt of a transaction.

Miscellaneous events are recorded asynchronously to incoming data transactions, for example E_PAGE_REQUEST.

7.2.1 Recording of events and conditions for writing to the Event queue

Events are delivered into an Event queue if the queue is "writable". The Event queue is writable when all of the following are true:

- The queue is enabled, through SMMU_(S_)CR0.EVENTQEN for the Security state of the queue.
- The queue is not full (see section 7.4 regarding overflow).
- No unacknowledged GERROR.EVENTQ_ABT_ERR condition exists for the queue.

When the queue is not writable, events not associated with stalled faults are silently discarded. In addition, a queue overflow condition is triggered when events are discarded because the queue is unwritable because it is full, see section 7.4.

Events caused by stalled transactions are not discarded. A stalled faulting transaction that has not recorded an event because the queue is unwritable has one of the following behaviors:

- If the stalled transaction is affected by a configuration or translation invalidation and CMD_SYNC, the transaction must be retried after the queue becomes writable again (non-full, enabled and without a queue abort condition). This either results in the transaction succeeding (because of a new configuration or translation) in which case no event is recorded for the transaction, or the generation of a new fault (reflecting new configuration or translation) which attempts to record an event into the queue, see section 4.6.3.
    - A transaction that was not affected by an invalidation or CMD_SYNC is permitted but not required to be retried in the same way as a transaction that is affected. The transaction might be retried at the point that the queue becomes writable.
- Alternatively, the transaction retries while the queue is unwritable:
    - If the retry translates successfully, the original event is permitted, but not required, to be recorded.
    - A fault that is encountered during a retry replaces any previous faults for the transaction. If the new fault causes the transaction to stall again, the new stall event is recorded when the queue becomes writable again. If the new fault causes the transaction to be terminated while the queue is still unwritable, the recording of the terminate event is lost.
- If the transaction is not retried, the original fault event record is recorded into the queue when it becomes writable.

Refer also to the event delivery effects of CMD_STALL_TERM (4.6.2), CMD_SYNC (4.6.3) and SMMUEN (6.3.9.6).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

An external abort on write of the Event queue might cause written event data to be lost, including stall event records. See section 7.2.2.

An event is permitted, but not required, to be recorded for a stalled faulting transaction when:

- The stalled transaction has early-retried and translated successfully before the SMMU has attempted to write out its event record, see section 3.12.2.2.

  Note: As the transaction has completed, there is no benefit in recording the original stall to software because software intervention is not required to progress the transaction.

Events arising from terminated faulting transactions commit to being recorded into the Event queue if it is writable. When EVENTQEN is transitioned to 0, committed events are written out and guaranteed to be visible by the time the update completes and all uncommitted events from terminated faulting transactions are discarded. If accesses to the Event queue aborted (see section 7.2.2 below), the condition is made visible in GERROR by the time the EVENTQEN update completes. See sections 6.3.9.4 and 6.3.9.6.

Note: There is no dependency required between EVENTQEN and visibility of any GERROR MSI that might arise from completion of outstanding queue writes that abort.

Some events are permitted to be recorded when SMMU_(S_)CR0.SMMUEN == 0 where explicitly stated in the event descriptions in this section. The remainder of events are related to the translation process and are not generated when translation is disabled with SMMUEN == 0.

Note: This means that SMMUEN == 0 does not imply EVENTQEN == 0.

7.2.2 Event queue access external abort

An external abort detected while accessing the Event queue, for example when writing a record, activates the SMMU_(S_)GERROR.EVENTQ_ABT_ERR Global Error. It is IMPLEMENTATION DEFINED as to whether the interconnect to the memory system can report transaction aborts. If EVENTQ_ABT_ERR is triggered, one or more events might have been lost, including stall fault event records.

An implementation is permitted to read any address within the bounds of the Event queue when the queue is enabled. In addition to writes of new records to the queue, such reads might also lead to an external abort.

The SMMU only writes to the Event queue when the queue is both enabled and writable, see section 7.2.1.

It is IMPLEMENTATION DEFINED whether an external abort on write is synchronous or asynchronous:

- If a queue write abort is synchronous, queue entry validity semantics are maintained so that all entries up to the SMMU_(S_)EVENTQ_PROD index are valid successfully-written event records. All outstanding queue writes are completed before the error is flagged in GERROR.EVENTQ_ABT_ERR. Where multiple outstanding record writes are performed simultaneously, records written at and beyond the queue location of the aborting record location are not visible to software even if they are written successfully, and are lost. The PROD index is not incremented for entries that caused a synchronous abort. In the scenario where a write of an event to an empty Event queue causes a synchronous abort, the PROD index is not incremented, the queue remains empty and the queue non-empty IRQ therefore is not triggered.
  - Note: Software can consume and process all valid entries in the Event queue.
- If a queue write abort is asynchronous, queue validity semantics are broken. The PROD index is permitted to be incremented for entries that caused an asynchronous abort. Software must assume all Event queue entries are invalid at the point of receiving the Global Error, and ARM strongly recommends

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

that the queue is made empty either by re-initialization or by the consumption or discarding of all (invalid) entries.

- Note: An IRQ might be observed for the activation of the SMMU_(S_)GERROR.EVENTQ_ABT_ERR condition in any order relative to an IRQ relating to the Event queue going non-empty as a result of the SMMU incrementing the PROD index for a queue entry that experiences an asynchronous external abort.

If the stalling fault model is implemented and enabled, software must terminate all stalling transactions that could be present in the SMMU by using CMD_STALL_TERM or a transition of SMMU_(S_)CR0.SMMUEN through 0.

See section 8.2 for similar PRI queue write abort behavior.

7.2.3 Secure and Non-secure Event queues

If two Security states are implemented, the Secure Event queue receives events relating to transactions from Secure streams. The Non-secure Event queue receives events relating to transactions from Non-secure streams. The Secure and Non-secure Event queues are independent. Non-secure faults or errors do not cause Secure event records and Secure faults or errors do not cause Non-secure event records.

7.3 Event records

Event records are 32 bytes in size, and are defined later in this section. Event records are recorded into the Event queue appropriate to the Security status of the StreamID causing the event, unless otherwise specified.

Common fields provided in events are:

- RnW: The Read/Write nature of the incoming transaction that led to the event
  - 0: Write
  - 1: Read
- PnU: Privileged/Unprivileged (post-STE override)
  - 0: Unprivileged
  - 1: Privileged
- InD: Instruction/Data (post-STE override)
  - 0: Data
  - 1: Instruction
- InputAddr: The 64-bit address input to the SMMU for the transaction that led to the event. The address includes any sign-extension that might occur before the SMMU input, as described in section 3.4.1. TBI does not affect the InputAddr field, which includes bits [63:56] in the same form as they were supplied to the SMMU.
  - Note: This field might be interpreted as a VA or an IPA depending on whether stage 1 translation is enabled.
- SSV: The SubstreamID validity flag
  - 0: No SubstreamID was provided with the transaction and the SubstreamID field is UNKNOWN.
  - 1: A SubstreamID was provided and the SubstreamID field is valid.
- S2: Stage of fault

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

    - 0: Stage 1 fault occurred
    - 1: Stage 2 fault occurred
- CLASS: The class of the operation that caused the fault
    - `0b00` CD CD fetch.
    - `0b01` TTD Stage 1 translation table fetch.
    - `0b10` IN Input address caused fault.
    - `0b11` Reserved.

Note: ARM recommends that software treats receipt of any event that is not architected here as a non-fatal occurrence.

The F_STE_FETCH, F_CD_FETCH and F_WALK_EABT events contain fields that report a fetch address, which is the address of a specific structure or descriptor that an aborting transaction was originally initiated to access. This STE, CD, or TTD address is as was calculated by a Stream table or CD table access or translation table walk.

Portions of event records that are not explicitly defined in this section are RES0.

7.3.1 Event record merging

Events originating from a stream are permitted to be merged when the stream has not been configured to explicitly inhibit merging with STE.MEV == 0. A merged event record is a single record written to the Event queue representing more than one occurrence of an event.

Two or more events are permitted but not required to be merged by an implementation when:

- The events are identical, or differ only as explicitly stated in event record definitions, and
- If the event type has a Stall field, Stall == 0. Events with a Stall parameter are never merged if Stall == 1.
- The events are not separated by a significant amount of time.
    - Note: The merging feature is intended to rate-limit events occurring at an unusually high frequency. ARM strongly recommends that an implementation writes separate records for events that do not occur in quick succession.

For the purposes of merging events, events are considered identical if all defined fields are the same except where explicitly indicated by an event definition.

The STE.MEV flag controls whether events resulting from a particular stream are merged. When STE.MEV == 0, events (other than listed below) are not merged; this can be useful for debug visibility where one transaction can be matched to one fault event. As STE.MEV is contained in an STE, it can only control the merging of events that are generated after a valid STE is located. The following events might occur before an STE is located, so might always be merged:

- F_UUT.
- C_BAD_STREAMID.
- F_STE_FETCH.
- C_BAD_STE.

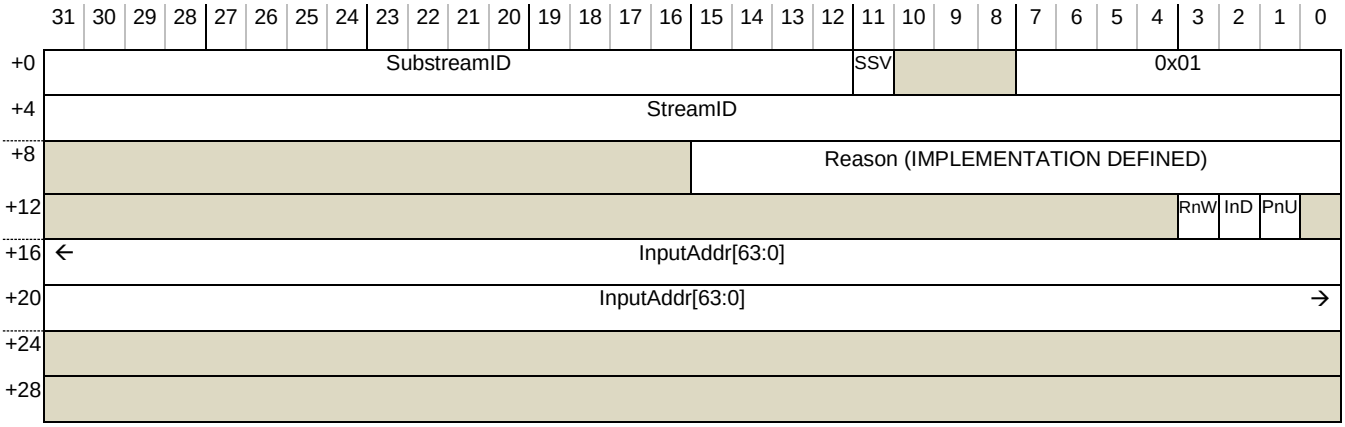
Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Hardware implementations are required to respect STE.MEV == 0, so that (other than the four events listed in this section) no events are merged. ARM recommends that software expects that event records might be merged even if STE.MEV == 0.

Note: In particular, software running in a virtual machine might set STE.MEV == 0 but a hypervisor might deem merging to remain valuable and cause it to remain enabled.

Note: An event that is recorded when STE.MEV == 1 and where Stall == 0 can therefore be interpreted as representing one or more transactions that faulted for the same reason.

7.3.2 F_UUT

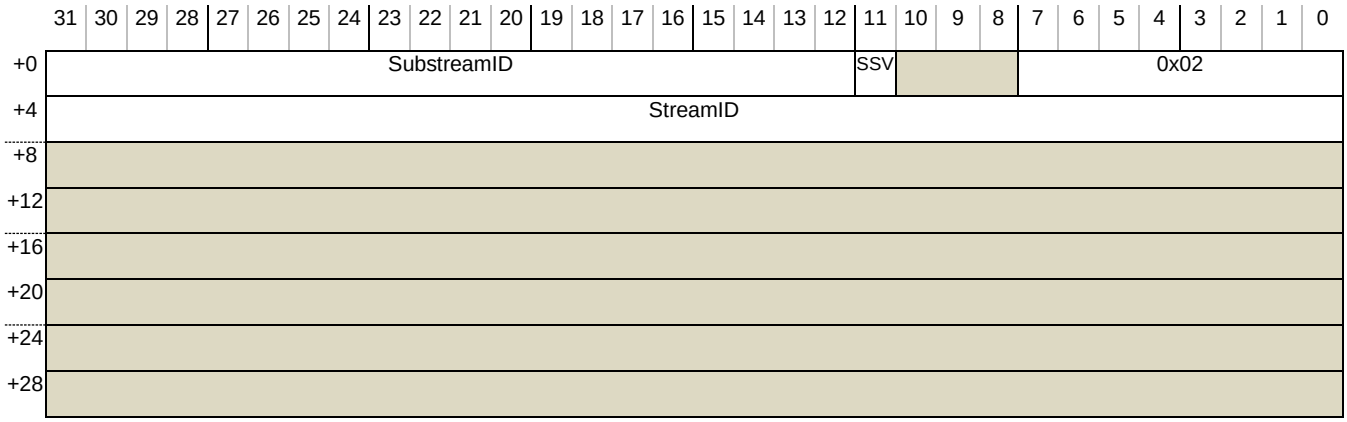


Event number	Reason
`0x01`	Unsupported Upstream Transaction. This event is caused for IMPLEMENTATION DEFINED reasons, depending on the bus system, a client device might express an unsupported or illegal transaction type. See section 16.7.1. An IMPLEMENTATION DEFINED cause is provided in Reason. The InD/PnU/NS attributes provided in this event are the incoming attributes, not the post-STE override versions.

This event is permitted to be recorded when SMMUEN == 0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

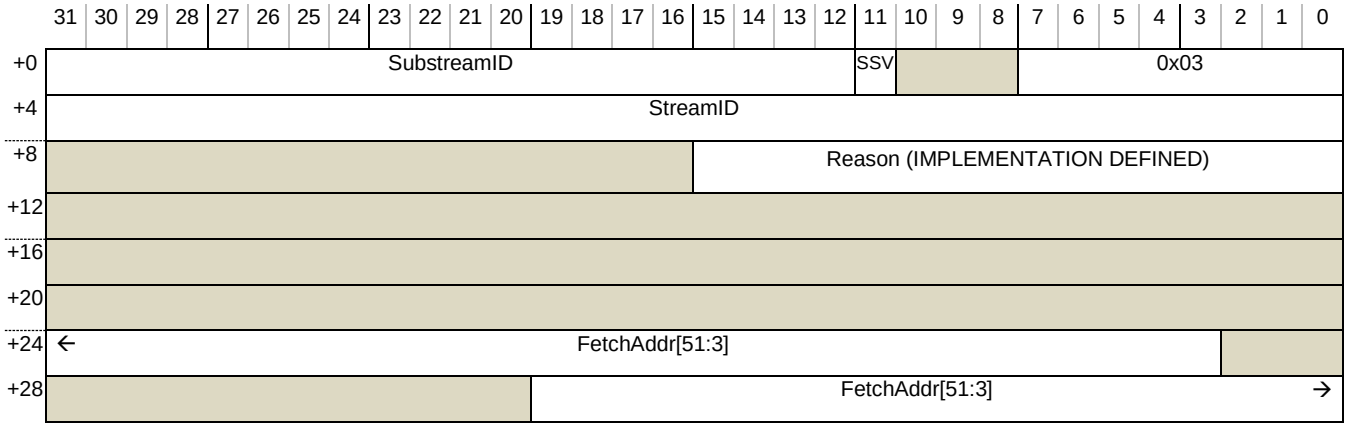
7.3.3 C_BAD_STREAMID



Event number	Reason
`0x02`	Transaction StreamID out of range. Recorded when SMMU_(S_)CR2.RECINVSID == 1 and any of the following conditions arises: 1) Incoming StreamID ≥$2^{\text{SMMU_(S_)STRTAB_BASE.LOG2SIZE}}$ 2) When SMMU_(S_)STRTAB_BASE.FMT == 2-level, given StreamID reaches a level 1 descriptor where: a) L1STD.Span == 0 b) L1STD.Span == Reserved c) L1STD.Span > SMMU_(S_)STRTAB_BASE_CFG.SPLIT + 1 d) The given StreamID[SPLIT - 1:0] ≥$2^{(\text{L1STD.Span-1})}$.

For out-of-range Secure StreamIDs, this event is recorded on the Secure Event queue and for Non-secure StreamIDs, the Non-secure Event queue. Each Security state might have a different number of StreamIDs (controlled by SMMU_STRTAB_BASE or SMMU_S_STRTAB_BASE).

7.3.4 F_STE_FETCH



Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

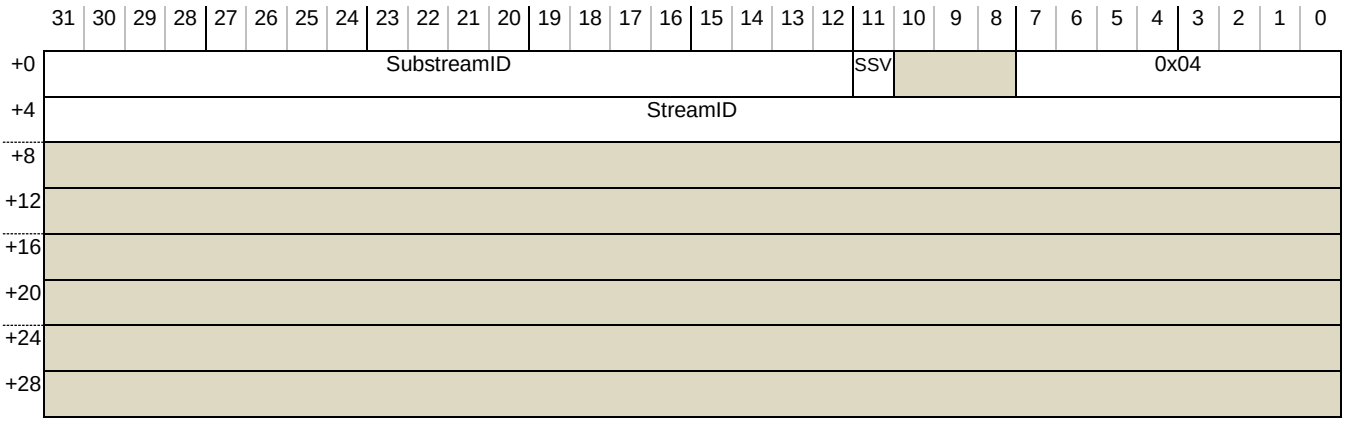
Event number	Reason
0x03	Fetch of STE caused external abort (access aborted by system/interconnect/slave, or consumed external error where RAS features available), or the address was out of range as described in 3.4.3. FetchAddr is the Physical Address used for the fetch. In SMMUv3.0, bits [47:OAS] are UNKNOWN. In SMMUv3.1, bits [52:OAS] are UNKNOWN. An IMPLEMENTATION DEFINED cause is provided in Reason.

Note: This event might be injected into a guest VM, as though from a virtual SMMU, when a hypervisor detects invalid guest configuration that would cause a guest STE fetch from an illegal IPA.

ARM recommends that an implementation with RAS features differentiates a failure that arose because of the consumption of an error from a failure caused by the use of an illegal address.

In SMMUv3.0, FetchAddr bits [51:48] are RES0.

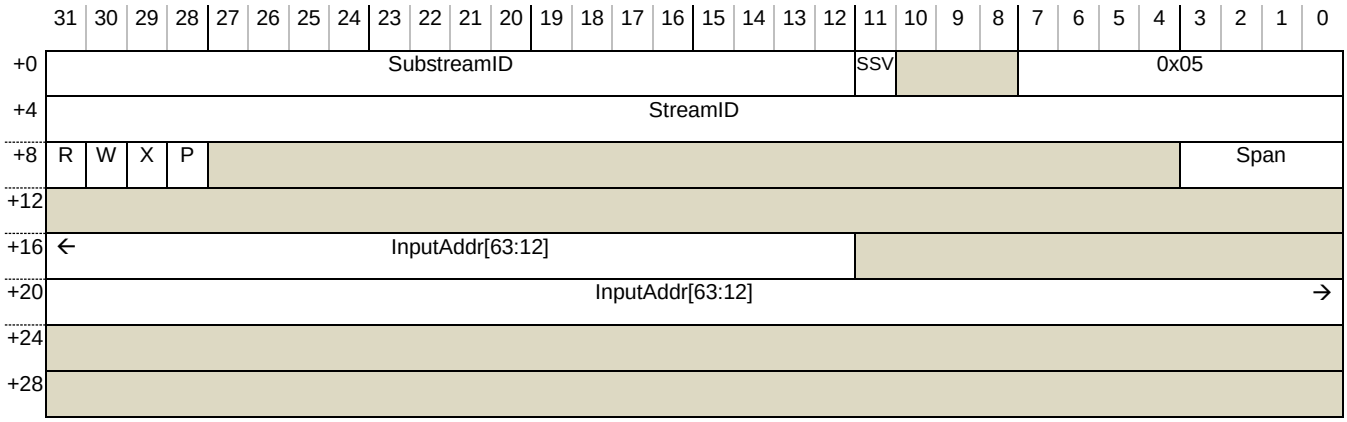
7.3.5 C_BAD_STE



Event number	Reason
0x04	Used STE invalid (V == 0, reserved field, incorrect configuration).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

7.3.6 F_BAD_ATS_TREQ



Event number	Reason
`0x05`	Address Translation Request disallowed for a StreamID and a PCIe ATS Translation Request received. R/W/X/PRIV are the Read/Write/Execute/Privilege permissions requested in the ATS Translation Request, before STE.{INSTCFG/PRIVCFG} overrides are applied: Write == !NW. For PCIe ATS, R (Read) is always 1 in a request. Span equals the number of STUs requested.

Reported in response to an ATS Translation Request in any of the following conditions:

- SMMU_CR0.SMMUEN == 0
- STE.V == 1 and effective STE.EATS == `0b00`
  - Note: See STE.EATS for details. The effective value of EATS is treated as `0b00` in some situations including if STE.Config == `0b100`, or STE is Secure.
- If it is possible for an implementation to observe a Secure ATS Translation Request, this event is recorded.

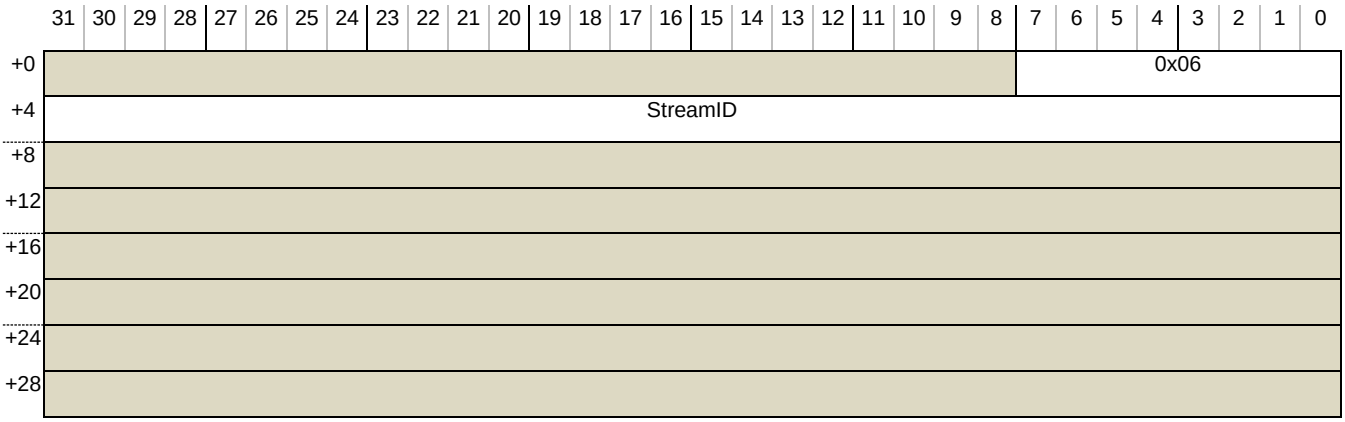
Note: This event is intended to provide visibility of situations where an ATS Translation Request is prohibited, but an ordinary transaction to the same address from the same StreamID or SubstreamID might complete successfully (where a failure of a TR might otherwise be difficult to debug by issuing an ordinary transaction).

Translation Requests do not cause other events (such as C_BAD_STE) to be recorded.

A UR response is made for an ATS Translation Request that causes this event.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

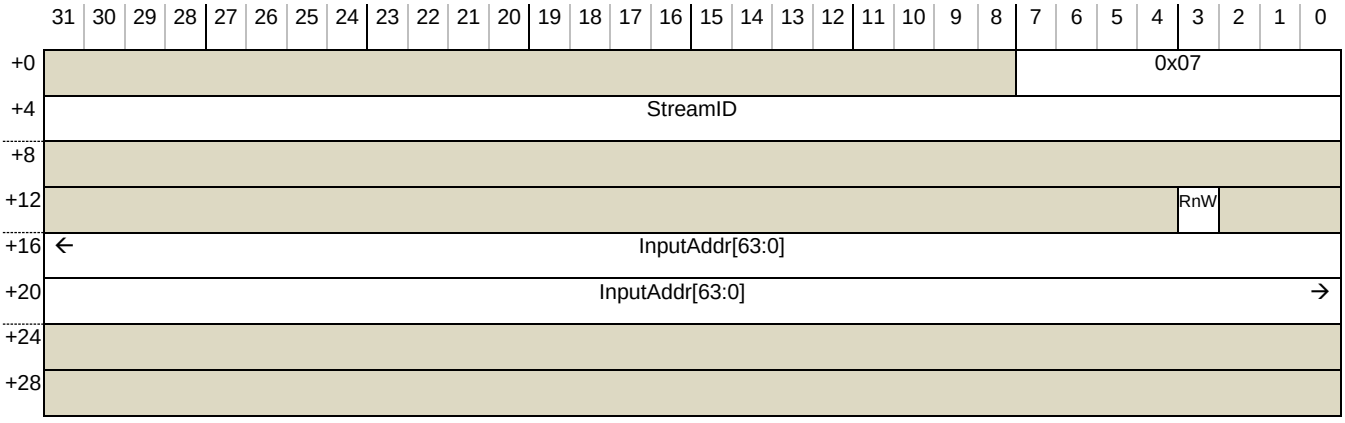
7.3.7 F_STREAM_DISABLED



Event number	Reason
`0x06`	The STE of a transaction marks non-substream transactions disabled (when STE.Config[0] == 1 and STE.S1CDMax > 0 and STE.S1DSS == `0b00`) and the transaction was presented without a SubstreamID. Or, a transaction was presented with SubstreamID 0 when STE.Config[0] == 1 and STE.S1CDMax > 0 and STE.S1DSS == `0b10` (CD 0 is reserved for non-substream traffic).

If STE.V == 1 and STE.Config == `0b000`, incoming traffic is terminated without recording an event.

7.3.8 F_TRANSL_FORBIDDEN



Event number	Reason
`0x07`	An incoming PCIe transaction is marked Translated but SMMU bypass is disallowed for this StreamID.

This event is permitted to be recorded when SMMUEN == 0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

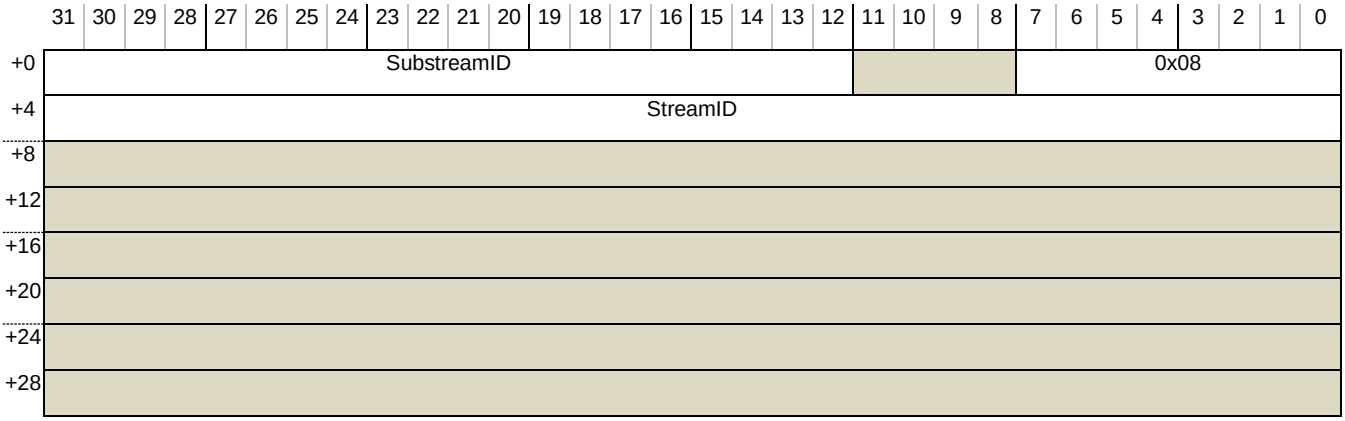
Reported in response to an ATS Translated transaction in any of the following conditions:

- SMMU_CR0.ATSCHK == 1, STE is valid and either:
  - STE disallows ATS in STE.EATS.
  - STE selects stage 1 & stage 2 bypass (STE.Config == `0b100`).
- StreamID is Secure (SEC_SID == 1).
- SMMUEN == 0.

Note: This event is intended to provide visibility of situations where an ATS Translated transaction is prohibited, but an ordinary (Untranslated) transaction from the same StreamID or SubstreamID might complete successfully, that is where behavior differs from an ordinary transaction because of the Translated nature of the transaction.

InputAddr[11:0] are IGNORED for the purposes of determining identical events for merging.

7.3.9 C_BAD_SUBSTREAMID



Event number	Reason
`0x08`	Incoming SubstreamID present and one of the following is true: • Stage 1 translation disabled (STE.Config[0] == 0) • Substreams disabled (STE.S1CDMax == 0) • SubstreamID ≥STE.S1CDMax • A two-level CD fetch encounters an L1CD structure with V == 0, or an L1CD structure that contains an L2Ptr that is out of range (see 3.4.3).

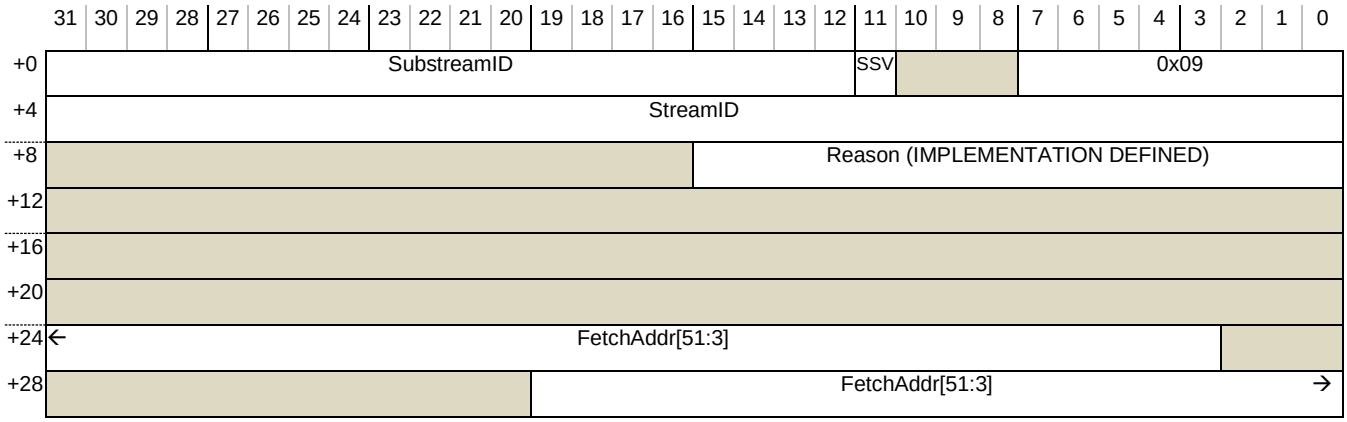
When caused by supply of a SubstreamID without stage 1 translation (STE.Config[0] == 0), this behavior arises independent of whether stage 2 translation is enabled.

When this event is generated because of an L1CD.L2Ptr that is out of range, the SubstreamID field indicates the index of the CD that was intended to be fetched when the erroneous L1CD.L2Ptr was used. In the case that an incoming transaction without a SubstreamID caused (through use of STE.S1DSS == `0b10`) an erroneous L1CD.L2Ptr to be encountered during fetch of the CD at index 0, the SubstreamID is recorded as 0. Otherwise, the SubstreamID that was input with the transaction is recorded.

Note: In this event, SubstreamID is always valid (there is no SSV qualifier).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

7.3.10 F_CD_FETCH

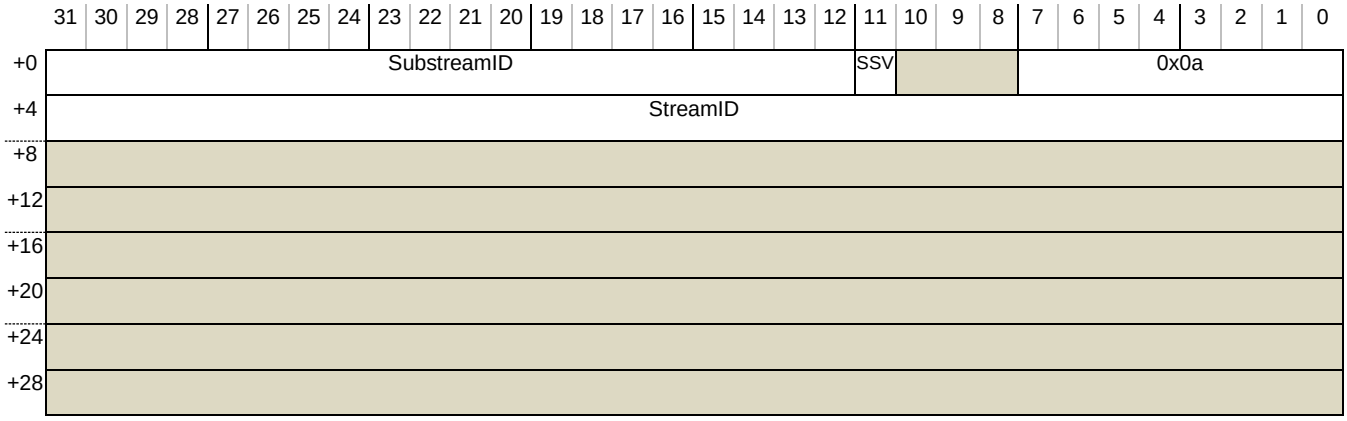


Event number	Reason
`0x09`	Fetch of CD caused external abort (access aborted by system/interconnect/slave, or consumed external error where RAS features available), or in SMMUv3.0 the address was out of range as described in 3.4.3. FetchAddr is the Physical Address used for the fetch. In SMMUv3.0, bits [47:OAS] are UNKNOWN. An IMPLEMENTATION DEFINED cause is provided in Reason.

Note: This event might be injected into a guest VM, as though from a virtual SMMU, when a hypervisor receives a stage 2 Translation-related fault indicating CD fetch as a cause (with CLASS == CD).

In SMMUv3.0, FetchAddr bits [51:48] are RES0.

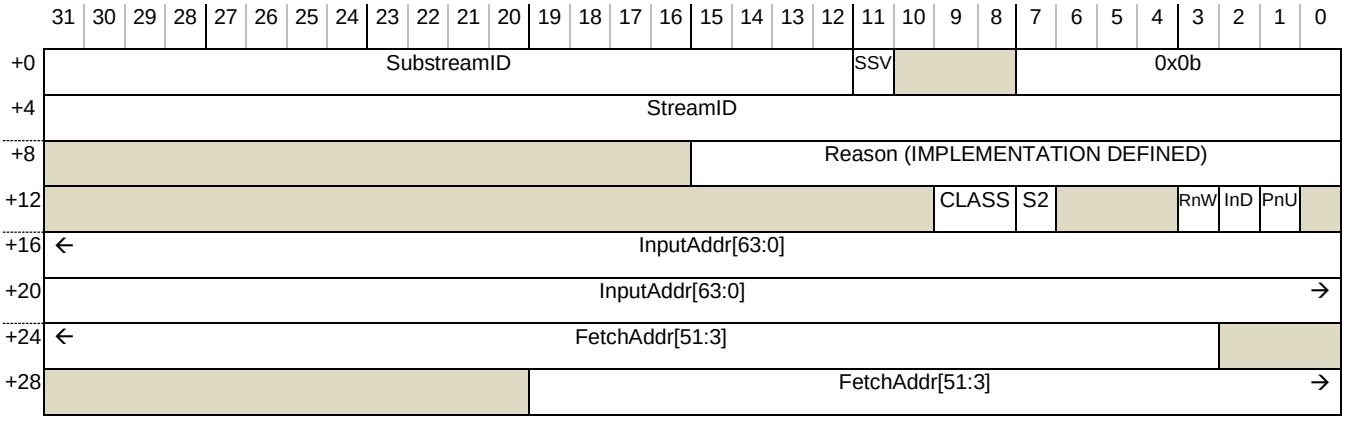
7.3.11 C_BAD_CD



Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Event number	Reason
0x0A	Fetched CD invalid (V == 0, or V == 1 but ILLEGAL configuration).

7.3.12 F_WALK_EABT



Event number	Reason

0x0B	An external abort occurred fetching (or updating) a translation table descriptor (access aborted by system/interconnect/slave, or consumed external error where RAS features available).
	
	The InD/PnU attributes are the post-STE override values. RnW pertains to the input transaction (InputAddr), not the access causing the abort. InD == 0 when RnW == 0 (see section 13.1.2).
	
	A stage 1-only table walk that encounters EABT on physical access to a descriptor is reported as S2 == 0 (stage 1), CLASS == TT. The SMMU input transaction address (InputAddr) and descriptor fetch PA (FetchAddr) are provided.
	Note: This behavior of CLASS == TT when S2 == 0 differs from other F_TRANSLATION, F_ADDR_SIZE, F_ACCESS, F_PERMISSION events relating to a stage 1 fault.
	
	A stage 2 table walk can encounter EABT accessing the physical address of a stage 2 descriptor, because of a:
	• Translation of an IPA for CD fetch
	  o S2 == 1 (stage 2), CLASS == CD
	  o The SMMU input address (InputAddr) and S2 descriptor fetch PA (FetchAddr) are provided
	• Translation of an IPA for Stage 1 descriptor fetch
	  o S2 == 1 (stage 2), CLASS == TT
	  o The SMMU input address (InputAddr) and S2 descriptor fetch PA (FetchAddr) are provided
	• Translation of an IPA after successful stage 1 translation (or, in stage 2-only

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Event number	Reason
	configuration, an input IPA) o S2 == 1 (stage 2), CLASS == IN (Input to stage) o The SMMU input address (InputAddr) and S2 descriptor fetch PA (FetchAddr) are provided. A stage 1 walk (nested with stage 2) can encounter EABT accessing the physical address of a stage 1 descriptors having successfully translated the IPA of the descriptor to a PA through stage 2: • S2 == 0 (stage 1), CLASS == TT • This is equivalent to the stage 1-only case. • The SMMU input address (InputAddr) and S1 descriptor fetch PA (FetchAddr) are provided. An IMPLEMENTATION DEFINED cause is provided in Reason. Note: This event occurs because of an incorrect configuration or, for systems supporting RAS features, consumption of an external error. A stage 1-only translation failing to walk a translation table might result from use of an incorrect or out of range PA in the table. When virtualization is in use, stage 2 translations would normally be fetched correctly. In addition, ARM recommends that hypervisor software does not allow a stage 1 walk to fetch a descriptor from an IPA that successfully translates to PA that causes an abort on access. When an AArch32 TTD fetch leads to an F_WALK_EABT on a system that has OAS < 40, a 40-bit descriptor address was truncated to the OAS to make the fetch. In this case, bits FetchAddr[47:OAS] are UNKNOWN.

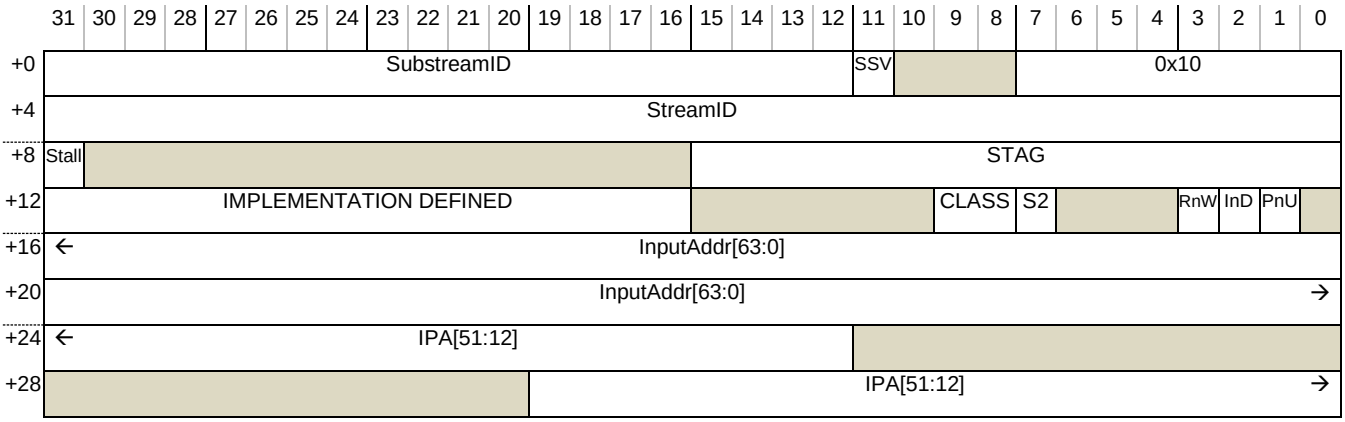
Note: To create the illusion of a guest VM having a stage 1-only virtual SMMU, this event can be injected into the guest if a stage 2 Translation, Access flag, Permission or Address Size fault occurs because of an access made by a stage 1 translation table walk (CLASS == TT).

InputAddr[11:0] are IGNORED for the purposes of determining identical events for merging.

In SMMUv3.0, FetchAddr bits [51:48] are RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

7.3.13 F_TRANSLATION



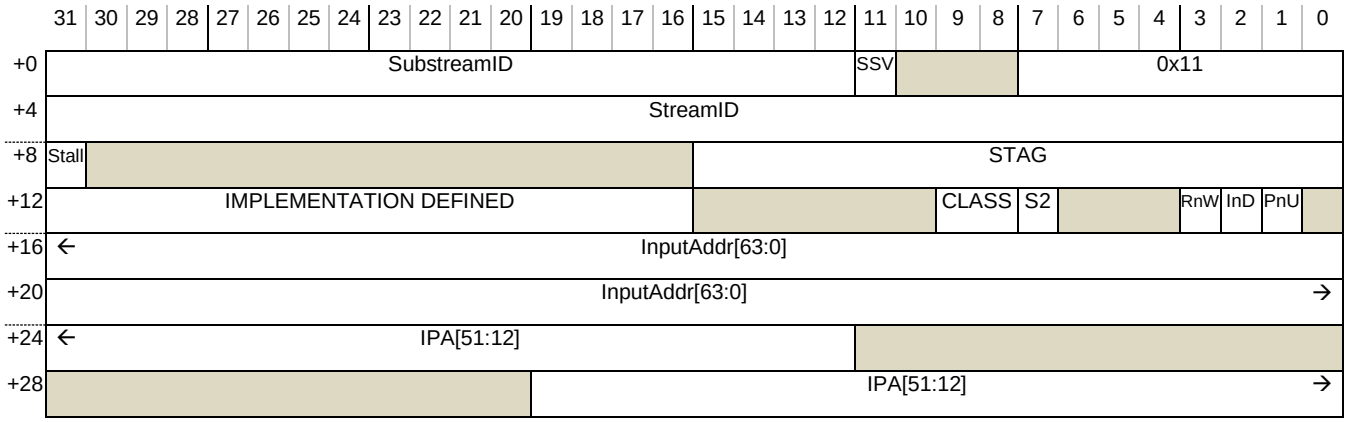
Event number	Reason
`0x10`	Translation fault: The address provided to a stage of translation failed the range check defined by TxSZ/SLx, the address was within a disabled TTBx, or a valid translation table descriptor was not found for the address. The RnW, InD and PnU fields provide the access attributes of the input transaction. The InD/PnU attributes are the post-STE override values. InD == 0 when RnW == 0. If fault occurs at stage 1, S2 == 0 and: • CLASS == IN, IPA is UNKNOWN. If fault occurs at stage 2, S2 == 1 and: • If fetching a CD, CLASS == CD, IPA is CD address • If walking stage 1 translation table, CLASS == TT, IPA is the stage 1 TTD address • If translating an IPA for a transaction (whether by input to stage 2-only configuration, or after successful stage 1 translation), CLASS == IN, and IPA is provided.

If Stall == 1, not merged. InputAddr[11:0] are IGNORED for the purposes of determining identical events for merging.

In SMMUv3.0, IPA bits [51:48] are RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

7.3.14 F_ADDR_SIZE



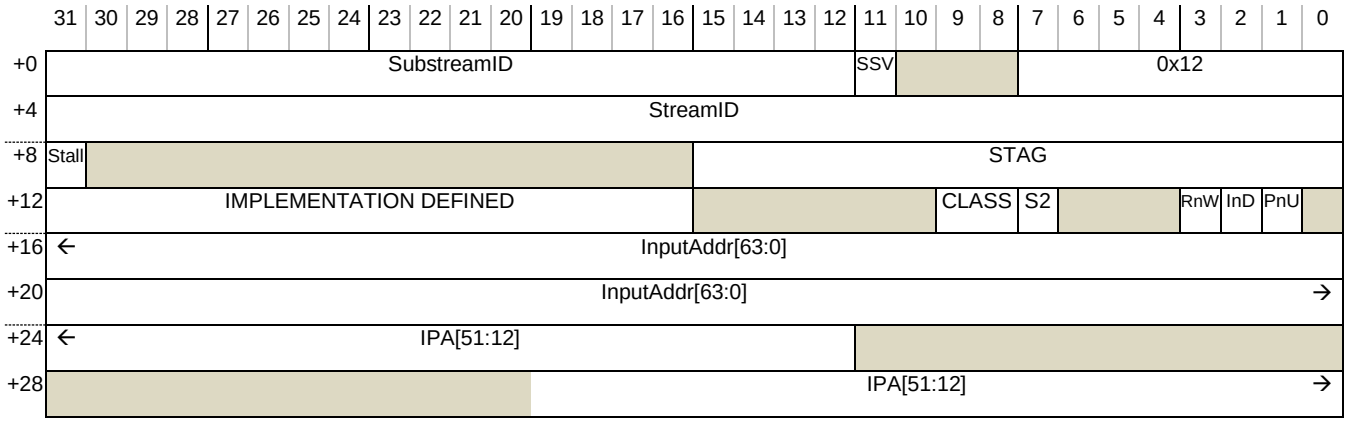
Event number	Reason
`0x11`	Address Size fault: Output address of a translation stage caused Address Size fault. When the stage performs translation, this fault occurs when an intermediate or leaf TTD outputs an address outside of the effective xPS associated with the translation table (see CD.IPS and STE.S2PS). This does not include a TTB address out of range before the TTW begins as, in this condition, the CD or STE is invalid which results in C_BAD_CD or C_BAD_STE. When stage 1 bypasses translation, this fault occurs when the output address (identical to the input address) is outside the address range implemented, see section 3.4. Refer to F_TRANSLATION for the definition of the CLASS, IPA and S2 fields. The RnW, InD and PnU fields provide the access attributes of the input transaction. The InD/PnU attributes are the post-STE override values. InD == 0 when RnW == 0. If caused by stage 1 translation bypass (because of stage 1 being disabled, unimplemented, or bypassed when STE.S1DSS == `0b01`), CLASS == IN and S2 == 0 (stage 2 bypass does not cause this fault).

If Stall == 1, not merged. InputAddr[11:0] are IGNORED for the purposes of determining identical events for merging.

In SMMUv3.0, IPA bits [51:48] are RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

7.3.15 F_ACCESS

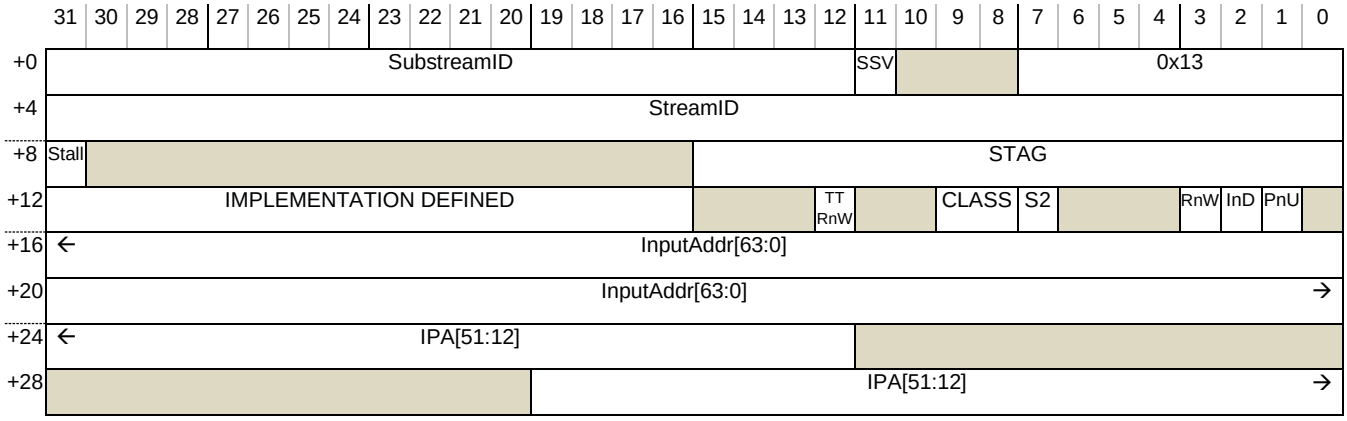


Event number	Reason
`0x12`	Access flag fault due to AF == 0 in a page or block TTD. If HTTU is supported and enabled, a TTD with AF == 0 is modified such that AF == 1 and this fault is not recorded. Refer to F_TRANSLATION for the definition of the CLASS, IPA and S2 fields. The RnW, InD and PnU fields provide the access attributes of the input transaction. The InD/PnU attributes are the post-STE override values. InD == 0 when RnW == 0.

If Stall == 1, not merged. InputAddr[11:0] are IGNORED for the purposes of determining identical events for merging.

In SMMUv3.0, IPA bits [51:48] are RES0.

7.3.16 F_PERMISSION



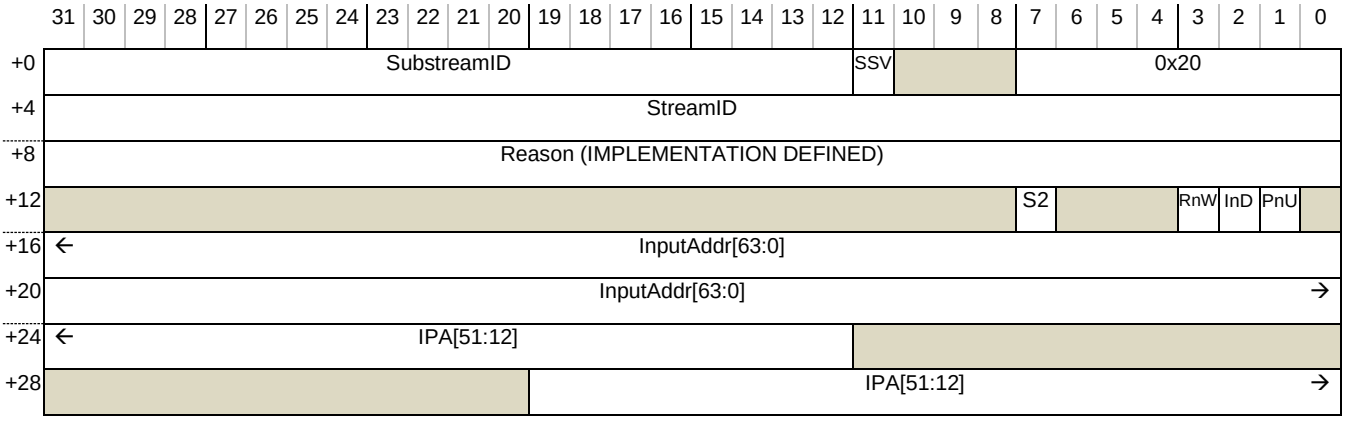
Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Event number	Reason
`0x13`	Permission fault occurred on page access. S2 indicates the stage at which fault occurred. The RnW, InD and PnU fields provide the access attributes of the input transaction. The InD/PnU attributes are the post-STE override values. InD == 0 when RnW == 0. Refer to F_TRANSLATION for the definition of the CLASS, IPA and S2 fields. The TTRnW field is valid when CLASS == TT, that is, a stage 1 TTD access causes a Permission fault at stage 2. TTRnW is UNKNOWN when CLASS is not TT. TTRnW indicates the read/write property that actually caused the fault as follows: 0 TTD Write caused S2 Permission fault. 1 TTD Read caused S2 Permission fault. Note: For example, an input read might end up causing a TTD write permissions fault at stage 2, when HTTU updates a stage 1 Access flag. Note: When CLASS == TT, the IPA field indicates the stage 1 TTD address. Note: ARM recommends that CLASS is used to determine the access properties causing the Permission fault. When CLASS == IN, RnW/InD/PnU were inappropriate for the access at the faulting stage. When CLASS == TT, the access is implicitly Data (regardless of the input property) and the faulting R/W property is given by TTRnW. When CLASS == CD, the access is implicitly Data and a read.

If Stall == 1, not merged. InputAddr[11:0] are IGNORED for the purposes of determining identical events for merging.

In SMMUv3.0, IPA bits [51:48] are RES0.

7.3.17 F_TLB_CONFLICT



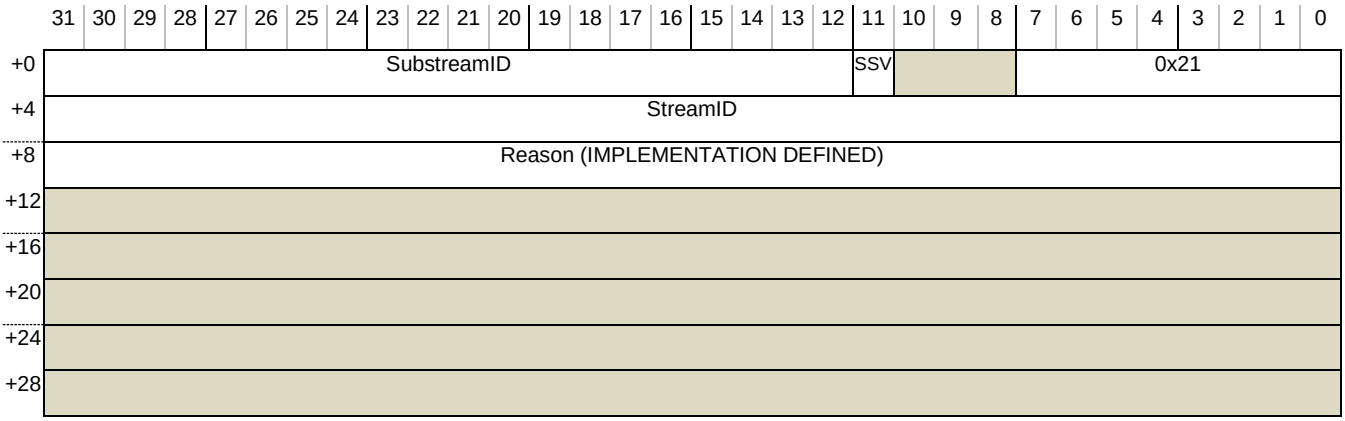
Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Event number	Reason
`0x20`	A TLB conflict occurred because of the transaction. The nature of this is IMPLEMENTATION DEFINED, defined by the TLB geometry of the implementation. The input address might not be the address that causes the issue, for example an IPA to PA TLB entry might experience the problem, so both addresses are provided. The RnW, InD and PnU fields provide the access attributes of the input transaction. The InD/PnU attributes are the post-STE override values. InD == 0 when RnW == 0. An IMPLEMENTATION DEFINED conflict reason is provided in Reason. This information might be specific to the TLB geometry of the implementation. IPA is valid when S2 == 1, otherwise it is UNKNOWN.

InputAddr[11:0] are IGNORED for the purposes of determining identical events for merging.

In SMMUv3.0, IPA bits [51:48] are RES0.

7.3.18 F_CFG_CONFLICT



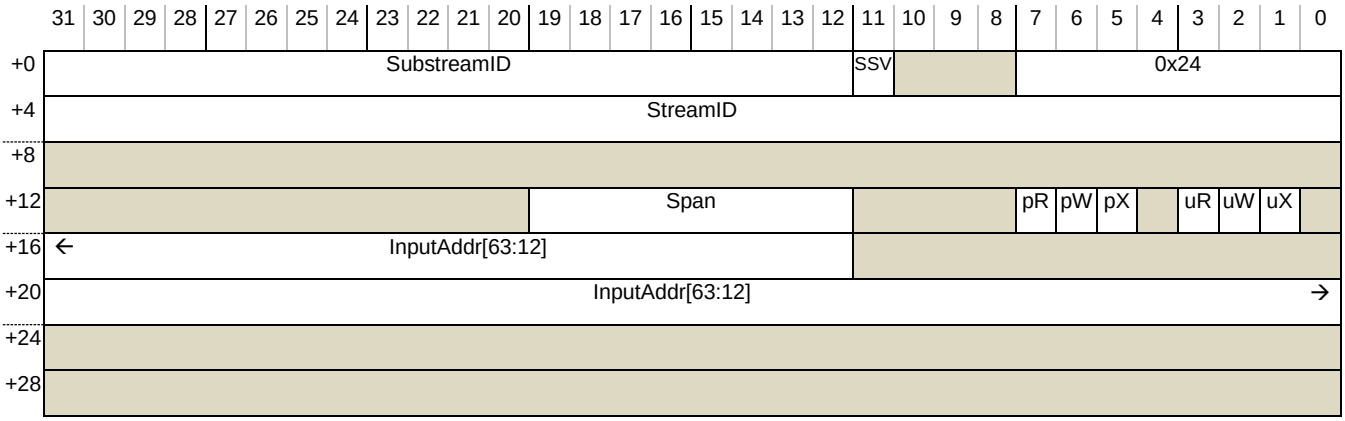
Event number	Reason
`0x21`	A configuration cache conflict occurred due to the transaction. The nature of this is IMPLEMENTATION DEFINED, defined by the configuration cache geometry of the implementation. The STE.CONT property of an STE cache entry causes an overlap of another STE that is not identical, as described in STE.CONT, and this has caused a multiple-match on lookup. An IMPLEMENTATION DEFINED conflict reason is provided in Reason. This information might be specific to the configuration cache geometry of the implementation.

It is not possible to cause this error by any CD misconfiguration.

Note: A guest VM cannot arrange configuration that would affect any other streams.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

7.3.19 E_PAGE_REQUEST



Event number	Reason
`0x24`	Speculative page request hint from a client device to system software, hinting that the given address span will be accessed by a device in the near future. The access is expected to require the read/write/execute properties provided for user/kernel privilege levels.

No response or action is required, but system software might use this hint to begin a (potentially long-running) page-in of the given address, to decrease the chance of a subsequent access from the device experiencing a page miss.

Flag field	*Access anticipated*
pR	Privileged Read
pW	Privileged Write
pX	Privileged Execute
uR	Unprivileged Read
uW	Unprivileged Write
uX	Unprivileged Execute

Span is encoded as a positive integer where the size of the intended access span in bytes is (Span*4096).

7.3.20 IMPDEF_EVENTn

Event number	Reason
`0xE0-EF`	IMPLEMENTATION DEFINED event allocation

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Notes: ARM recommends that software treats receipt of these events as a non-fatal occurrence even if the exact nature of the event is not recognised.

All other event numbers are Reserved.

Events with a Stall parameter (F_TRANSLATION, F_ADDR_SIZE, F_ACCESS, F_PERMISSION) indicate whether a transaction is stalling as a result of the event. When a transaction causes any other event, the transaction is terminated and an abort returned to the client. If a transaction is stalled, the STAG parameter provides a token used to uniquely identify the transaction.

Note: The StreamID and STAG must be provided to a subsequent CMD_RESUME. Every stall event must have either one associated CMD_RESUME or a CMD_STALL_TERM.

A configuration error always terminates the instigating transaction with an abort to the device.

Multiple addresses might be reported in one event record:

- InputAddr is the address input with the transaction to the SMMU.
- IPA is the post-stage 1 address of the transaction, valid only if the fault occurred at stage 2
- Fetch PA is the physical address accessed causing the fault to be raised

The input address is the 64-bit address presented to the SMMU, see section 3.4.1.

7.3.21 Event queue record priorities

Events arising from an ordinary incoming transaction are recorded in the following order, listed from the first that might occur when a transaction is processed to the last that might occur before the transaction is deemed successful, unless otherwise specified. This means that if any event occurs, events prior to that event in this list cannot have occurred:

1. C_BAD_STREAMID
2. F_STE_FETCH
3. C_BAD_STE
4. C_BAD_SUBSTREAMID
5. F_STREAM_DISABLED
6. Faults from translation (stage 2, for fetch of CD)
7. F_CD_FETCH
8. C_BAD_CD
9. Faults from translation (stage 1 or stage 2, for data access of a transaction)

Faults from translation can occur because of a fetch of a CD or stage 1 translation requiring a stage 2 translation, or the final translation for the given transaction address.

A single stage of translation table walk might experience faults in this order:

1. F_TRANSLATION (for input address outside range defined by TxSZ/SL0, or EPDx == 1)
2. For each level of translation table walked:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

    a. F_WALK_EABT (on fetch of TTD)
    b. F_TRANSLATION (from fetched TTD)
    c. F_ADDR_SIZE (for output of TTD, indicating next TTD or walk output address)
3. F_ACCESS (On the final-level TTD)
4. F_PERMISSION (On the final-level TTD)

A nested Stage 1 + Stage 2 translation table walk might experience faults in this order:

1. F_TRANSLATION (For stage 1 input address outside range defined by stage 1 TxSZ/SL0, or EPDx == 1)
2. For each level of stage 1 translation table walked, the stage 2 translation for the TTD of the stage 1s IPA might experience:
    a. F_TRANSLATION (For stage 2 input address outside range defined by stage 2 T0SZ/SL0)
    b. F_WALK_EABT (For stage 2 TTD fetch)
    c. F_TRANSLATION (From fetched stage 2 TTD)
    d. F_ADDR_SIZE (For output of stage 2 TTD, indicating next TTD or walk output address)
    e. F_ACCESS (On the final-level stage 2 TTD)
    f. F_PERMISSION (On the final-level stage 2 TTD)
3. For each level of stage 1 translation table walked, after the PA of the TTD is determined the following might occur:
    a. F_WALK_EABT (For stage 1 TTD fetch)
    b. F_TRANSLATION (From fetched stage 1 TTD)
    c. F_ADDR_SIZE (For output of stage 1 TTD)
4. For the final level stage 1 TTD, the following might then occur:
    a. F_ACCESS
    b. F_PERMISSION
5. The IPA relating to the input VA has been determined and permissions have been checked, but the following might occur for each level of the stage 2 translation of that final IPA:
    a. F_TRANSLATION (For stage 2 input address outside range defined by stage 2 T0SZ/SL0)
    b. F_WALK_EABT (For stage 2 TTD fetch)
    c. F_TRANSLATION (From fetched stage 2 TTD)
    d. F_ADDR_SIZE (For output of stage 2 TTD, indicating next TTD or walk output address)
    e. F_ACCESS (On the final-level stage 2 TTD)
    f. F_PERMISSION (On the final-level stage 2 TTD)

Note: F_TLB_CONFLICT, F_CFG_CONFLICT and F_UUT are raised with implementation specific prioritisation.

The flow of events that an incoming ordinary transaction might raise can be characterised as:

- An unsupported transaction fault from the client device might be raised on input.
- Configuration is fetched, in the order of STE then (if relevant) CD:
    - If a CD is fetched, it uses stage 2 translations to do so, if stage 2 is enabled.
    - Stage 2 walk occurs, which might experience an external abort on fetch.
    - Stage 2 page fault on the CD address might occur when the walk completes.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

    - External abort on CD fetch might occur.
- When a CD is successfully fetched, start walking stage 1 TT. Stage 2 page faults (or external aborts) might arise from walking stage 1, in the same way as for the initial CD fetch
- When a S1 translation is determined, stage 1 might fault.
- Stage 2 walk occurs for a successful output of stage 1 translation, and this might also fault.

ATS Translation Requests do not lead to the same set of possible errors as, in general, ATS translation errors are reported to the requesting device using the ATS Translation Response rather than to software through the Event queue. The only Event queue record that can be generated is F_BAD_ATS_TREQ, which can be raised at several points in the translation process.

ATS Translated transactions also differ from ordinary transactions and one of the following behaviors can occur:

- F_TRANSL_FORBIDDEN can be raised on input when ATS is disabled/invalid (see section 7.3.8 for reasons).
- Otherwise, ATS Translated transactions that locate correct configuration and then encounter stage 2 translation (through STE.EATS == `0b10` configuration) can raise stage 2 Translation-related faults.
- Otherwise, any other error on receipt of an ATS Translated transaction (for example, an invalid STE) causes the transaction to be silently terminated.

See section 15 for more details on the translation procedure and possible events at each step.

7.4 Event queue overflow

An SMMU output queue is full if the consumer can observe that there are no free entries for the producer to add new entries to, see section 3.5.2. The Event queue is full if it can be observed that SMMU_(S_)EVENTQ_PROD.WR == SMMU_(S_)EVENTQ_CONS.RD and WRAP bits differ.

Note: If the SMMU can observe that a write it made for a new queue entry is visible in memory, but PROD.WR has not yet been updated to pass the position of the new entry, the new entry is not yet visible to software in terms of the queue visibility semantics.

Event queue overflow occurs when the Event queue is enabled (EVENTQEN == 1), the queue is full, and a pending event record is discarded because the queue is full. An Event queue overflow condition is entered to indicate that one or more event records have been lost:

- Fault information.
- Configuration errors.

It is not possible for an external agent to observe an Event queue overflow condition if it could not first observe the Event queue being full.

Note: An implementation must therefore prevent the overflow condition from being observable by software before an update to the PROD index is observable.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

A configuration error record or fault record from a terminated transaction is discarded when the SMMU attempts to insert it into a full Event queue. However, a fault record from a stalled transaction is not discarded and an event is reported for the stalled transaction when the queue is next writable. In addition, as stall fault records are not discarded, stall fault records do not cause an overflow condition. Only discarded events cause overflow.

When EVENTQEN == 0, events are not delivered and non-stall event records might be discarded but do not cause an overflow condition.

The Event queue overflow condition is present when:

SMMU_EVENTQ_PROD.OVFLG != SMMU_EVENTQ_CONS.OVACKFLG

When an overflow occurs, the SMMU toggles the SMMU_EVENTQ_PROD.OVFLG flag if the overflow condition is not already present.

Note: ARM recommends that whenever software reads SMMU_EVENTQ_PROD, it checks OVFLG against its own copy. If different, the overflow condition arose. ARM recommends that software then updates its copy with the value of OVFLG, and processes the Event queue entries as quickly as possible, updating SMMU_EVENTQ_CONS in order to move on the RD pointer but also to acknowledge receipt of the overflow condition by setting OVACKFLG to the same value read from OVFLG.

Note: In terms of delivering events, a queue in an unacknowledged overflow state does not behave any differently to normal queue state. If software were to consume events and free space but leave overflow unacknowledged, new events could be recorded. Because a second overflow condition cannot be indicated, a subsequent queue overflow would be invisible therefore ARM expects software to acknowledge overflow upon the first consumption of events after the condition arose.

7.5 Global error recording

Global Errors pertaining to a programming interface are reported into the appropriate SMMU_(S_)GERROR register instead of into the memory-based Event queue.

SMMU_(S_)GERROR provides a one-bit flag for each of the following error conditions:

Error flag	Meaning
CMDQ_ERR	Command queue error
EVENTQ_ABT_ERR	Event queue access aborted Delivery into the Event queue stops when an abort is flagged, see section 7.2.
PRIQ_ABT_ERR	PRI queue access aborted Delivery into the PRI queue stops when an abort is flagged, see section 8.2.
MSI_CMDQ_ABT_ERR	CMD_SYNC MSI write aborted
MSI_EVENTQ_ABT_ERR	Event queue MSI write aborted
MSI_PRIQ_ABT_ERR	PRI queue MSI write aborted (Non-secure GERROR only)
MSI_GERROR_ABT_ERR	GERROR MSI write aborted

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Error flag	Meaning
SFM_ERR	SMMU entered Service Failure Mode This error is common to both SMMU_GERROR and SMMU_S_GERROR, so that both secure and Non-secure software are aware that the SMMU has entered SFM.

When an error condition is triggered, the error is activated by toggling the corresponding flag in GERROR. In some cases, SMMU behavior changes while the error is active:

- Commands are not consumed from the Command queue while CMDQ_ERR is active.

The presence of a Global Error is acknowledged by the agent controlling the SMMU by toggling the equivalent field in the GERRORN register.

An error is active when: SMMU_(S_)GERROR[x] != SMMU_(S_)GERRORN[x]

The SMMU does not toggle bit[x] if the error is already active. If one or more new errors occur while a previous error of the same type is active, the new error is not logged.

Note: This handshake avoids a race condition that could otherwise lead to loss of error events. It follows that an error flag indicates that one or more of the indicated errors has occurred since the last acknowledgement.

It is IMPLEMENTATION DEFINED whether the interconnect to the memory system of an implementation returns abort completions for writes, which lead to *_ABT_ERR global errors.

7.5.1 GERROR interrupt notification

A GERROR interrupt is triggered when the SMMU activates an error, with the exception of the following fields:

- MSI_GERROR_ABT_ERR
    - Note: This error signifies that a prior attempt to raise a GERROR MSI had been aborted, and sending another to the same address might cause a second abort.

The GERROR interrupt is triggered only when SMMU_(S_)IRQ_CTRL.GERROR_IRQEN == 1. When MSIs are used, GERROR MSI notification is configured using SMMU_(S_)GERROR_IRQ_CFG{0,1,2}. If multiple errors activate at the same time, GERROR interrupts are permitted to be coalesced.

Where the architecture guarantees that a GERROR flag update is visible before the completion of an action that might have triggered an error, completion of such an action is not required to depend on the GERROR interrupt having been made visible.

Note: For example, an IRQ enable update from 1 to 0 does not complete until the abort of a prior MSI is recorded as GERROR.MSI_n_ABT_ERR, but the update is permitted to complete before the GERROR interrupt is triggered.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

8 Page request queue

The message format arriving on the PRI queue in response to an incoming PCIe PRI message closely follows the PCIe packet format. The field names have been generalised for consistency with terminology elsewhere in this specification. The system must ensure that the mapping between StreamID and RequesterID or Root Complex is identical whether used in either incoming or outgoing directions for PRI, ATS or normal DMA traffic.

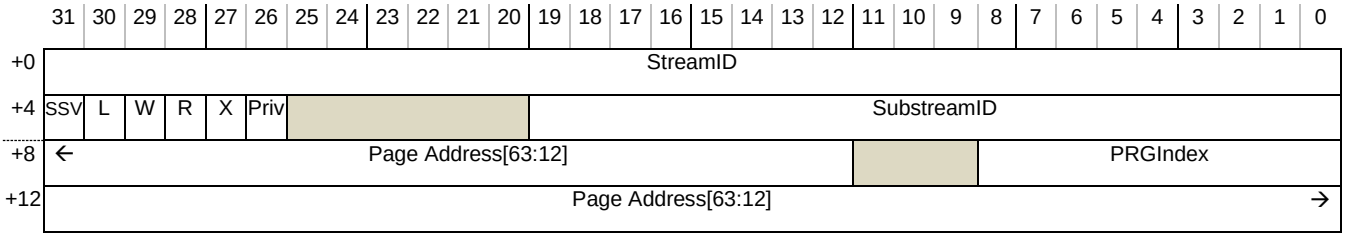


Figure 18: PRI queue entry format

- StreamID[31:0]
  - When used with ATS/PRI, StreamID[15:0] must equal PCIe RequesterID[15:0]. Bits StreamID[n:16] might indicate different Root Complex sources where more than 16 bits of StreamID are implemented.
- SSV: Substream valid
  - When 1, the Page Request was issued with a PASID TLP prefix.
  - When 0, the Page Request was not issued with a PASID. This value is also 0 when Substreams are not supported by the SMMU. When SSV == 0, the X and Priv bits are both 0.
- SubstreamID[19:0]
  - This value is UNKNOWN when SSV == 0.
- Page address[63:12]
- PRGIndex[8:0]
- Last, W, R, X, Priv: Last/Write/Read/eXecute/Privileged:
  - These bits encode requested page permissions
  - 'Last' is set on the last request in a PRG
  - The PCIe 'Stop PASID' marker message delimits Page Request message groups as a request with LWR == `0b100`. Stop Markers have a PASID, that is, SSV == 1. A message that arrives without a PASID and that has LWR == `0b100` is not a Stop Marker, has SSV == 0, and is treated as a PRI Page Request.

Several related PRI Page Requests (PPRs) might arrive in batches, named Page Request Groups (PRGs). Several pages might be requested in a PRG and all arrive with the same PRGIndex value. The last page request in a PRG is marked by the Endpoint as Last == 1 and the PRI queue can be configured to send an interrupt when Last == 1 is received (see 6.3.36). Member requests of a PRG are not guaranteed to be contiguous in the PRI queue as requests of more than one PRG might be interleaved. PPRs that are members of the PRG are not guaranteed to appear in the PRI queue in the order that they were issued from an endpoint, with the exception of the Last == 1 entry which is not re-ordered with respect to prior PPRs.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The SMMU is not required to track and verify that, for a given PRGIndex value, all entries received in a PRG have identical PASID TLP prefixes.

The CMD_PRI_RESP command causes a PRI response to be sent to an endpoint and takes a success or failure parameter to inform the endpoint of the overall status of the PRG. When Substreams (PASIDs in this context) are supported and this command is used with a valid SubstreamID (SSV == 1), a PASID tag is present on the response.

Note: ARM expects this command to be issued after all PPRs in a PRG have been processed, including the Last == 1 entry.

Note: PCIe PASID re-use requires a message indicating all PPRs that have been issued with a particular PASID have now been seen. This is signalled by the 'Stop PASID' marker.

Note: The PRI queue does not overflow with correct software usage and endpoint credit management, because software grants credits to each device (using PCI configuration space) which sum to the amount of space in the PRI queue minus an allowance for any 'Stop PASID' markers that are required. The device can only issue as many outstanding PPRs as it has credits for.

The SMMU freely accepts incoming PRI messages and puts them in the PRI queue, if space is free and the queue is enabled and not in an error condition. If the queue is full a programming error (or protocol failure by PCIe endpoint) has occurred.

Note: A device is intended to always receive a response to a PRG, whether positive or negative.

Note: A CMD_PRI_RESP returns a credit to the Endpoint that could cause another PPR. This must be considered when calculating credit capability of a queue. If the number of credits granted to devices equals the queue capacity, a PPR must be consumed from the queue (freeing an entry) prior to issuing a CMD_PRI_RESP for that PPR.

The SMMU supports a maximum PRI queue size of 2^{19} entries.

Note: When a guest VM manages a PRI queue for page request service from one or more devices assigned to that guest, it is up to the guest OS to allocate and size its local PRI queue. It will subsequently give credits to the devices to allow them to issue PPRs, using configuration space accesses. In this scenario, the hypervisor manages the actual SMMU PRI queue and the sum of the credits given to all clients of that queue must not be greater than the size of the queue. For safety, ARM expects that a hypervisor will trap the write of a credit value of a guest to a device, and substitute its own allocation if necessary. This would ensure that the total number of credits cannot exceed the PRI queue size, even if a guest OS requests a larger number of credits to be used. The hypervisor is also expected to choose an appropriate PRI queue size which will approximate the sum of credits reasonably expected to be used by all guest VMs using the PRI facilities of an SMMU.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

8.1 PRI queue overflow

The PRI queue enters an overflow condition when the following conditions apply:

- The PRI queue is enabled, as indicated by the effective value of SMMU_CR0.PRIQEN == 1.
- One or more PRI messages are received by the SMMU while the PRI queue is full, as determined by the general queue semantics.

The SMMU indicates that this has happened by toggling the SMMU_PRIQ_PROD.OVFLG flag, provided that an overflow condition is not already present. The overflow condition is acknowledged by writing the SMMU_PRIQ_CONS.OVACKFLG flag to the same value as OVFLG.

An active overflow condition inhibits new entries from being written to the PRI queue.

Note: On detection of overflow, ARM recommends that software processes the received PRI queue entries as quickly as possible, and follows this with a single write of SMMU_PRIQ_CONS to simultaneously move on the RD pointer and acknowledge receipt of the overflow condition by setting OVACKFLG to the value read from OVFLG.

When an overflow condition is active, incoming PRI messages are discarded and:

- An automatic PRG Response is generated for Page Request messages having Last == 1, including the PRGI of the PPR that caused the overflow condition to become active. This response is returned to the endpoint that originated the PPR. The ResponseCode is defined below.

  Note: This behavior occurs only for a Page Request message and does not occur when a Stop Marker is discarded in an overflow condition.
- PPRs having Last == 0 are silently ignored.

Note: The auto-response behavior can maintain safe operation when PCIe endpoints fail to fully adhere to the PRI credit mechanism.

Note: Although in an overflow condition the system software will not have addressed the cause of the PPR, on receipt of the automatic successful response ARM expects that the device will gracefully attempts an ATS request and PRI request again. On subsequent requests, system software might have had time to process the PRI queue contents and free space for the retries.

Note: Auto-responses might be sent because of an overflow condition, or for other reasons. See section 8.2.

When the SMMU supports PASIDs, an automatic PRG Response that is sent in response to a PRI queue overflow condition has the following properties:

- If the associated Page Request message did not have a PASID, the response has no PASID and the ResponseCode field of the response is Success (`0b0000`). SMMU_IDR3.PPS and STE.PPAR do not affect this case.
- If the associated Page Request message had a PASID:
  - If SMMU_IDR3.PPS == 1, then the response has the same PASID and the ResponseCode is Success (`0b0000`). STE.PPAR is not checked and its value is ignored.
  - If SMMU_IDR3.PPS == 0, the STE.PPAR field associated with the StreamID of the PPR is checked. The PASID depends on the associated STE:
    - If the associated STE is valid, the response has the same PASID if STE.PPAR == 1 and it has no PASID if STE.PPAR == 0. In both cases, ResponseCode is Success (`0b0000`).
    - If the associated STE is inaccessible or invalid, such that STE.PPAR cannot be checked, no PASID is used and the ResponseCode is Failure (`0b1111`) . This occurs if SMMU_CR0.SMMUEN == 0, the StreamID is out of range of the Stream table, an External Abort was encountered during the STE fetch, or the STE is ILLEGAL.

    Note: For a normal transaction, the same conditions lead to C_BAD_STREAMID, F_STE_FETCH, and C_BAD_STE, respectively.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When PASIDs are not supported, an automatic response does not have a PASID prefix and the STE.PPAR field is not used. In this case, an implementation is permitted, but not required, to attempt to locate a valid STE and return either ResponseCode == Success or Failure in the manner described in this chapter even though STE.PPAR is not required. If the implementation does not perform this STE check, the ResponseCode == Success.

Note: A PRI-capable endpoint advertises whether it expects a PASID prefix to be present on a PRG response returned to it in response to a Page Request made with a PASID prefix, through the PRG Response PASID Required flag in the PRI Status register of the endpoint. ARM expects the STE.PPAR field associated with an endpoint to be programmed to the same value as this flag in the endpoint.

Stop PASID markers that arrive in an overflow condition are discarded and no auto-response is generated.

Note: The PCIe PRI specifications [3] [9] do not consider Stop PASID markers to be credited in the same way as PPRs. If endpoints disrespect Page Request allocation and credits, Stop PASID markers cannot be guaranteed to be visible to the system, given this rule. Software must use other means to determine PASID stop status in the presence of non-compliant PCIe endpoints.

8.1.1 Recovery procedure

At the time that an overflow is detected, the PRI queue might contain several groups of page requests that are legitimate, but might contain one or more groups of requests that have been truncated so that their 'Last' event was among those lost when overflow occurred, and these groups have had a 'Success' auto-response sent when the 'Last' event was discarded.

Note: To re-synchronise and regain normal processing of page requests, one possible method is:

- Process the entire queue: Entries are read from the SMMU_PRIQ_CONS.RD index up to the SMMU_PRIQ_PROD.WR index, sending Page Request Group Response commands using the Command queue as appropriate, when 'Last' entries are encountered.
- The PRI queue might contain the start of groups of page requests whose Last requests were lost, having been received and auto-responded to after overflow. A group must be ignored by software if no Last event has been found for it up to the point that the SMMU last wrote (the SMMU_PRIQ_PROD.WR index).
- Update SMMU_PRIQ_CONS.RD, including OVACKFLG, to clear the overflow condition and mark the entire queue as empty or processed.

Note: After an overflow condition has been resolved, a PRGIndex value might be used in PPRs that later become visible, associated with a semantically different PRG to that of a PRG with entries in the pre-overflow PRI queue. This would occur when the queue enters overflow state before the Last PPR of the initial PRG can be recorded. The Last PPR has a successful auto-response sent, meaning the endpoint is free to re-use the PRGIndex value.

8.2 Miscellaneous

An external abort detected while accessing the PRI queue, for example when writing a record, activates the SMMU_GERROR.PRIQ_ABT_ERR Global Error. If PRIQ_ABT_ERR becomes active, one or more PRI records might have been lost. The behavior of PRIQ_ABT_ERR relates to entries written to the PRI queue as the behavior of EVENTQ_ABT_ERR relates to entries written to the Event queue, as described in section 7.2.2. It is IMPLEMENTATION DEFINED whether an external abort that is triggered by a write to the PRI queue is synchronous or

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

asynchronous. When the write of a PRI queue entry results in a synchronous external abort, an automatic PRG Response with ResponseCode == 0b1111 is generated for the associated PPR. When the write of a PRI queue entry results in an asynchronous external abort, one or more queue entries might have been lost and no automatic response is generated for the PPRs that are associated with these entries.

An implementation is permitted to read any address within the bounds of the PRI queue when the queue is enabled (that is, when the effective value of SMMU_CR0.PRIQEN == 1). In addition to writes of new records to the queue, such reads might also lead to an external abort.

A PPR received from a Secure stream is discarded, is not recorded into the PRI queue and results in an automatic Response Message having ResponseCode == `0b1111`.

If an active GERROR.PRIQ_ABT_ERR error condition exists, or if the effective value of SMMU_CR0.PRIQEN == 0 (SMMU_CR0.SMMUEN == 0 implies PRIQEN == 0) all of the following apply:

- No entries are written to the PRI queue.
- All incoming PPRs cause an automatic PRG Response having ResponseCode == `0b1111` ('Response Failure') and the messages are discarded.

Note: Incoming PRI Page Requests are not affected by SMMU_CR0.ATSCHK or STE.EATS configuration.

The SMMU does not generate responses to Stop Marker messages.

Automatically-generated PRG responses have the following properties:

- Same StreamID as the PPR that triggered the response so that the message is returned to the originating RequesterID.
- Same Page Request Group Index as the PPR that triggered the response.
- Response Code depends on the cause of auto-generated response, see this section and sections 8.1 and 8.3.
- No PASID prefix is used on responses that were auto-generated because the effective value of PRIQEN == 0 or PRIQ_ABT_ERR occurred.
- A PASID prefix is used if all of the following are true:
    - The response was auto-generated because of PRI queue overflow.
    - The SMMU supports substreams.
    - SMMU_IDR3.PPS == 1, or SMMU_IDR3.PPS == 0 and STE.PPAR == 1.
    - The response will not have ResponseCode == `0b1111`.
    - The PPR that triggered the auto-response had a PASID prefix.
- If a PASID prefix is used, its PASID value matches that of the PPR that triggered the response.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

8.3 PRG Response Message codes

ResponseCode	Status	Returned for
0b0000	Success	CMD_PRI_RESP with Resp == Success • Software has paged in all pages successfully. This response is returned for an auto-response on overflow where no PASID was supplied on the associated request, or SMMU_IDR3.PPS == 1, or no error is encountered in checking STE.PPAR.
0b0001	Invalid Request	CMD_PRI_RESP with Resp == InvalidRequest • Software failed to page-in all pages. It encountered one or more pages that do not exist or have insufficient privileges
0b1111	Response Failure	CMD_PRI_RESP with Resp == ResponseFailure • Software has determined that ATS is disabled for the stream, or some other (non-mapping-related) permanent failure. This response is returned for an auto-response on overflow where SMMU_IDR3.PPS == 0 and the associated request was supplied with a PASID and the SMMU is unable to locate a valid STE when checking STE.PPAR. This response is also returned for all incoming PPRs when any of the following conditions exist: • The effective value of PRIQEN == 0 (which might be because SMMU_CR0.SMMUEN == 0), • A GERROR.PRIQ_ABT_ERR error is active. • The PPR is received from a Secure stream.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

9 Address Translation Operations

The SMMU optionally supports a software-accessible Address Translation Operations (ATOS) facility. The presence of this facility is reported by SMMU_IDR0.ATOS. The ATOS facility allows software to perform an address lookup to determine the output address that a transaction would take, given an input address, stream, substream and access properties. The requested lookup can be the full end-to-end, with all configured stages, or only a part of the configuration for the stream. The result of the request is either the output address or a fault status code, if a translation is unable to complete. Unlike transactions, ATOS requests are not affected by fault configuration bits in the STE or CD, and do not record fault events or stall.

If ATOS is supported, an optional virtual ATOS (VATOS) page might also be supported. This is reported by SMMU_IDR0.VATOS. The VATOS page can be mapped through to a chosen software entity so that it might perform a limited set of stage 1-only ATOS lookups directly. The VATOS interface can make stage 1-only ATOS lookups for stage 1 and stage 2 nested configurations, or for stage 1-only configurations. The VATOS interface can only make lookups for configurations associated with the NS-EL1 StreamWorld.

Note: ARM expects VATOS to be used with stage 1 and stage 2 nested configurations.

When ATOS is supported, a group of SMMU_GATOS_SID, SMMU_GATOS_ADDR and SMMU_GATOS_PAR registers are available in the memory map. When SMMU_S_IDR1.SECURE_IMPL == 1, a second group is present in SMMU_S_GATOS_SID, SMMU_S_GATOS_ADDR, SMMU_S_GATOS_PAR. If VATOS is supported, a third group, SMMU_VATOS_SID, SMMU_VATOS_ADDR and SMMU_VATOS_PAR, are present in a distinct VATOS page, see section 6.

Each group of registers might perform one lookup at a time, but all implemented sets of registers might perform independent lookups simultaneously. Access to a register that affects ATOS behavior only affects the group that is accessed, and programming of the registers (whether correct or incorrect) does not affect other groups.

Apart from differences in scope the groups of registers are equivalent and are referred to generically using ATOS_ prefixes in this chapter. The difference in scope between the ATOS registers is that the VATOS registers can only access stage 1 translations, the GATOS registers cannot access Secure stream information, and the S_GATOS registers can access Secure stream information.

The ATOS registers might perform address lookups with the following properties as though a transaction were received through a given StreamID and SubstreamID:

- Read/Write.
- Unprivileged/Privileged.
- Instruction/Data.

The types of translations available are:

- VA to PA for a stage 1 and stage 2 configuration.
- VA to IPA for a stage 1 of a stage 1 and stage 2 configuration.
- VA to IPA or PA for a stage 1-only configuration.
- IPA to PA for a configuration with stage 2.

The lookup procedure is:

1. Ensure the ATOS interface is idle (ATOS_CTRL.RUN == 0) and claimed for exclusive use in case several agents share the interface

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

2. Ensure ATOS_SID identifies the StreamID and SubstreamID that will be used for the lookup
3. Write the source address and lookup action to ATOS_ADDR.
4. Ensure the prior register writes are observed by the SMMU before the following write.
5. Write ATOS_CTRL.RUN == 1 to initiate the lookup.
6. Poll ATOS_CTRL.RUN; the SMMU clears the RUN flag when the result is ready.
7. The result appears in ATOS_PAR. This might be a result address or a fault code, determined in the same way as for an ordinary input transaction.

The translation process for an ATOS request is not required to begin immediately when ATOS_CTRL.RUN is written to 1, and will begin in finite time. ARM expects that any delay will be small.

An ATOS translation interacts with configuration and TLB invalidation in the same way as a translation that is performed for a transaction. The result of an ATOS translation that completes after an invalidation completes is not based on any stale data that was targeted by the invalidation. See section 3.21 for more information.

Note: A CMD_SYNC depends upon the visibility of HTTU updates performed by completed ATOS translations.

The completion of an invalidation operation is permitted, but not required, to depend on either of the following:

- The completion of an ATOS translation that has been requested by writing RUN to 1 but has not begun before the invalidation completion operation was observed.
- The completion of an ATOS translation that is unaffected by the invalidation.

The domain of the lookup, in Secure, Non-secure, hypervisor, EL3 or Monitor terms, is determined by the stream configuration. Translations from a Secure stream are only returned through the S_ATOS interface. A Non-secure stream might be queried by either interface. The S_ATOS interface contains a SSEC flag to indicate whether the requested StreamID is to be treated as Secure or Non-secure.

The lookup action provided in the write to ATOS_ADDR determines the read/write, instruction/data and privilege attributes of the request.

The treatment of the input address that is provided by ATOS_ADDR is governed by the same rules that affect an ordinary translation, in terms of range checks according to TxSZ, IPS, and PS SMMU properties. See section 3.4. ATOS_ADDR.ADDR provides an input 64-bit address.

Note: As the full address is provided, a translation lookup could fail with a Translation fault because it is outside the configured input range for the initial stage or stages of lookup. This includes consideration of the Top Byte Ignore (TBI) configuration.

ATOS_ADDR indicates the read/write, instruction/data and privileged/unprivileged properties of the translation request. The permissions model used by an ATOS translation is the same as is used for an ordinary transaction that has the same effective read/write, instruction/data, privileged/unprivileged attributes after STE overrides are taken into account for the transaction.

Note: The ATOS permission attributes are not affected by the STE.INSTCFG or STE.PRIVCFG overrides.

In the case where an ATOS operation explicitly performs translation at only one stage when a stream is configured for stage 1 and stage 2 translation, only the permission configuration that is applicable to the requested translation stage is used.

Note: This means that CD-based permission controls such as PAN, UWXN and WXN are not used when an ATOS translation does not translate using stage 1. When stage 1 is in use, all of the CD-based permission

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

controls are used by ATOS, in the same way as would be done for an ordinary transaction. These are CD.PAN, CD.UWXN, CD.WXN and CD.HAD{0,1}.

When ATOS_ADDR.HTTUI == 0, ATOS operations perform HTTU when supported by the SMMU and configured for any of the translation regimes used to perform the lookup, in the same way as would occur for a read or write transaction to the same address of the same StreamID/SubstreamID, including fetch of CD and Stage 1 TTDs through stage 2 translations.

When ATOS_ADDR.HTTUI == 1, the request inhibits HTTU and the behavior is as follows:

- When an Access flag update is enabled for a stage of translation (STE.S2HA == 1 or CD.HA == 1), the SMMU is permitted but not required to set the Access flags in TTDs for the requested address. If AF == 0, then the ATOS translation behaves as though AF == 1, even if the Access flag is not updated. If a translation-related fault or external abort occurs as a result of performing the Access flag update, it is IMPLEMENTATION DEFINED whether the ATOS operation reports the fault or whether the operation continues as though the Access flag was successfully updated.
- When update of the Dirty state of a page is enabled for a stage of translation (STE.S2HD == 1 or CD.HD == 1) a TTD with DBM == 1, and without write permission at the required privilege level because of the value of AP[2] == 1 or S2AP[1] == 0, is considered to be writable by the ATOS mechanism but the AP[2]/S2AP[1] bits are not updated.

Setting ATOS_ADDR.HTTUI == 1 has no effect when updates to both the Access flag and Dirty state of the page are disabled.

Note: The behavior of an ATOS operation with HTTUI == 1 matches that of address translation operations in the PE, with respect to AF and Dirty state of the page permissions.

Note: If HTTU is inhibited in order to translate without changing the page table state, this inhibition includes stage 2 translation tables that back the stage 1 page table in nested configurations.

When HTTU is enabled at stage 2 in a nested stage 1 and stage 2 configuration, and ATOS performs a stage 1-only VA to IPA translation and HTTU is enabled for ATOS, the SMMU does not mark the final IPA's page as dirty and is permitted but not required to mark the final IPA's page as accessed.

ATOS can only be used when SMMUEN == 1 for the relevant programming interface, that is SMMU_CR0.SMMUEN == 1 for SMMU_GATOS_*, SMMU_VATOS_* or SMMU_S_CR0.SMMUEN for SMMU_S_GATOS_*.

An ATOS request is permitted to use and insert cached configuration structures and translations, consistent with any caches that are provided for transaction translation. Regardless of the value of ATOS_ADDR.HTTUI , an ATOS request that uses a configuration that has HTTU enabled, that is a configuration with STE.S2HA == 1, CD.HA == 1, STE.S2HD == 1, or CD.HD == 1, does not insert or update TLB entries that would be marked as Accessed or Dirty when their associated TTDs in memory are not.

When SMMU_S_IDR1.SECURE_IMPL == 1, the SMMU does not allow:

- Secure information to be used to form a Non-secure ATOS_PAR result
- A Non-secure ATOS request to affect SMMU_S_GATOS_* register values, including PAR, or active Secure ATOS translation requests.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

9.1 Register usage

9.1.1 ATOS_CTRL

Software writes ATOS_CTRL.RUN to 1 to request a translation. The bit is cleared back to 0 by the SMMU when ATOS_PAR has been updated with the result.

Software must not write any register in the ATOS group when ATOS_CTRL.RUN == 1.

An ATOS request can only be made when the SMMU_(S_)CR0.SMMUEN field that is relevant to the ATOS register group in use is 1. The ATOS registers must not be written if SMMUEN == 0.

SMMUEN must not be cleared while one or more ATOS requests are in progress for the programming interface, otherwise ongoing requests might be terminated. ATOS_CTRL.RUN is cleared by an Update of SMMUEN to 0.

Note: An ATOS request that is made through the Secure programming interface for a Non-secure StreamID might be affected by Non-secure software simultaneously clearing SMMU_CR0.SMMUEN from another PE, so must accept spurious ATOS translation failures which might result.

See section 6.3.37 for more information on ATOS_CTRL.RUN and alteration of the SMMUEN flags.

9.1.2 ATOS_SID

The StreamID and optional SubstreamID are written to ATOS_SID.STREAMID, ATOS_SID.SUBSTREAMID and ATOS_SID.SSID_VALID in preparation for a subsequent ATOS lookup. When ATOS_CTRL.RUN is written to 1, the value of this register is used as input to the process.

The SMMU_S_GATOS_SID register contains a SSEC flag that selects between Secure and Non-secure StreamID lookup for an ATOS request that is issued from the Secure interface.

Note: Non-secure ATOS interfaces have no SSEC flag, so they can only look up Non-secure StreamIDs.

9.1.3 ATOS_ADDR

The address to be queried is written to ATOS_ADDR.ADDR.

The type of lookup that is required is written to the TYPE field:

ATOS_ADDR.TYPE	Lookup scope	Notes
`0b00`	Reserved	Generates ATOS error INV_REQ

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

ATOS_ADDR.TYPE	Lookup scope	Notes
`0b01`	Stage 1 (Look up IPA/PA from VA)	Only format available when SMMU supports only stage 1 translation. Only format available to VATOS interface. Note: It is permissible to make a stage 1 request for a StreamID with either a stage 1-only or a stage 1 and 2 configuration. Generates ATOS error INV_REQ when SMMU does not support stage 1 translation. Generates ATOS error INV_STAGE when stage 1 is supported but not configured to translate for the given StreamID. Note: This does not require a stage 1-only configuration for the StreamID. This lookup returns the top half of a nested stage 1 and stage 2 configuration.
`0b10`	Stage 2 (Look up PA from IPA)	Generates ATOS error INV_REQ when used from the SMMU_VATOS_ADDR interface, or when the SMMU does not support stage 2 translation. Generates ATOS error INV_STAGE when stage 2 is supported but not configured to translate for the given StreamID. Generates ATOS error INV_STAGE when issued from the Secure ATOS interface with SMMU_S_GATOS_SID.SSEC == 1. Generates ATOS error INV_REQ if ATOS_SID.SSID_VALID (because no stage 1 translation is requested). Note: This does not require a stage 2-only configuration for the StreamID. This lookup returns the bottom half of a nested stage 1 and stage 2 configuration.
`0b11`	Stage 1 and stage 2 (Look up PA from VA)	Generates ATOS error INV_REQ when used from SMMU_VATOS_ADDR interface, or when the SMMU does not support both stage 1 and stage 2. Generates ATOS error INV_STAGE when stage 1 and stage 2 are supported but not both configured to translate for the given StreamID. Generates ATOS error INV_STAGE when issued from the Secure ATOS interface with SMMU_S_GATOS_SID.SSEC == 1.

When TYPE == `0b01` or `0b11` (requesting stage 1), the generation of INV_STAGE is determined from whether stage 1 is configured to translate by STE.Config[2:0] and is not caused by a STE.S1DSS == `0b01` configuration.

A configuration with STE.Config[2:0] == `0b0xx` or `0b100` results in INV_STAGE.

A request with ATOS_SID.SSID_VALID == 0 on a stream with STE.S1DSS == `0b01` leads to one of the following:

- A stage 1 lookup (TYPE == `0b01`) behaves according to stage 1-bypass, which returns the address provided in ATOS_ADDR.ADDR without translation, or a stage 1 fault.

  If no fault occurs and a result is returned, the result contains the following properties:

  - ATOS_PAR.ADDR is the input address plus a translation size encoded in conjunction with ATOS_PAR.Size. The translation size is permitted to be any value between the minimum implemented granule size and the IAS.
  - ATOS_PAR.ATTR and ATOS_PAR.SH are IMPLEMENTATION DEFINED.
  - SMMU_S_GATOS_PAR.NS is IMPLEMENTATION DEFINED.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: It is possible for an Address Size fault to result from stage 1 in bypass.

- A stage 1 and stage 2 lookup (TYPE == `0b11`) behaves according to stage 1-bypass, and stage 2-translates, returning the address provided in ATOS_ADDR.ADDR translated at stage 2, or a stage 1 fault or a stage 2 fault.

The required read/write, instruction/data and privileged/unprivileged properties of the ATOS request are written to ATOS_ADDR.RnW, ATOS_ADDR.InD and ATOS_ADDR.PnU, respectively. These attributes are not affected by the STE.INSTCFG or STE.PRIVCFG overrides.

9.1.4 ATOS_PAR

ATOS_PAR contains the result of the translation request, which provides either a translated address and attributes or an error result.

The content of ATOS_PAR is only valid when a translation has been both initiated and completed through the sequence of the corresponding RUN bit having been set to 1 by software, and then cleared to 0 by the SMMU.

The content of ATOS_PAR is UNKNOWN when:

- The SMMU_(S_)CR0.SMMUEN relevant to the ATOS register group is 0.
- RUN == 1.

If ATOS_PAR.FAULT == 0, the translation was successful and ATOS_PAR.ADDR returns the translated address, in addition to the page size, and other fields provide the determined access attributes. See section 6.3.40 for details of the fields.

The attributes that are returned in ATOS_PAR.ATTR and ATOS_PAR.SH are those that are determined from the translation for a stage 1-only or a stage 2-only request or for a stage 1 and stage 2 request. See section 13.1.5 for definition of attribute combination. Because TTD.SH only encodes a Shareability for Normal memory, a Shareability of OSH is always provided in ATOS_PAR.SH when a Device memory type is determined from this procedure. The attributes do not include STE overrides and are permitted to reflect the attributes that are stored in TLB entries. The attributes that are stored in the TLB entries might be an IMPLEMENTATION DEFINED subset of the architectural TTD/MAIR attributes/types, see section 6.3.40.

If ATOS_PAR.FAULT == 1, an error or fault occurred during translation. In this case, ATOS_PAR.FAULTCODE reports whether there was:

- An error in invocation, which is INV_REQ or INV_STAGE, as described in section 9.1.3.
- A fault or error during translation, corresponding to a standard SMMU event type, for example C_BAD_STE or F_TRANSLATION.
- An internal error. For example, a RAS error that caused the translation to terminate, or the case where SMMUEN was cleared to 0, thereby terminating the translation.

Other fields in ATOS_PAR determine the following:

- REASON. This field determines whether the issue was encountered at stage 1, or at stage 2 because of the incoming IPA, or because of fetching a stage 1 translation or a CD through an IPA that caused a stage 2 fault.
- FADDR. This field determines the IPA that caused a stage 2 fault.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The validity of the REASON and FADDR fields changes depending on the type of request that is made and the kind of fault encountered. In the following table:

- **TRF** means any of the Translation Related Faults (F_TRANSLATION, F_ADDR_SIZE, F_ACCESS, F_PERMISSION).
- **MISC** means all other faults, with the exception of F_WALK_EABT, F_CD_FETCH, INV_REQ and INV_STAGE. MISC represents: INTERNAL_ERR, C_BAD_STREAMID, F_STE_FETCH, C_BAD_STE, F_STREAM_DISABLED, C_BAD_SUBSTREAMID, C_BAD_CD, F_TLB_CONFLICT, F_CFG_CONFLICT.

ATOS_ADDR. TYPE	Possible FAULTCODE values	Possible PAR. REASON values	FADDR contents	Notes
Any	INV_REQ, INV_STAGE	`0b00`	0	Error in invocation. See section 9.1.3.
`0b01` (stage 1) on stage 1-only stream	F_WALK_EABT	`0b00` (S1)	0	TTD fetch abort
	F_CD_FETCH			CD fetch abort
	TRF, MISC			
`0b01` (stage 1) on stage 1 and 2 stream	F_WALK_EABT	`0b00` (S1)	0	Stage 1 TTD fetch abort (actual bus abort)
	F_CD_FETCH			CD fetch abort (actual bus abort)
	F_CD_FETCH, if Stage 2 F_WALK_EABT or stage 2 TRF is caused by CD fetch.			CD fetch abort (synthetic, because of a stage 2 translation failure)
	F_WALK_EABT, if stage 2 F_WALK_EABT or stage 2 TRF is caused by stage 1 TTD fetch.			Stage 1 TTD fetch abort (synthetic, because of a stage 2 translation failure)
	TRF, MISC			Configuration or miscellaneous fault, or stage 1 TRF. Note: The IPA that is determined from a stage 1 translation is not further translated to a PA, therefore stage 2 TRFs on an illegal IPA do not occur.
`0b10` (stage 2)	F_ADDR_SIZE	`0b11` (S2, IN)	0	Stage 2 Address Size fault.
	F_ADDR_SIZE	`0b00` (S1)	0	Stage 1 Address Size fault. It is IMPLEMENTATION DEFINED whether an SMMUv3.0

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

ATOS_ADDR. TYPE	Possible FAULTCODE values	Possible PAR. REASON values	FADDR contents	Notes
				implementation reports this fault as REASON == `0b00` or `0b11`.
	F_TRANSLATION, F_ACCESS, F_PERMISSION, F_WALK_EABT, MISC.	`0b11` (S2, IN)	0	
`0b11` (stage 1 and stage 2)	TRF	`0b00` (S1)	0	Stage 1 translation-related fault
		`0b01` (S2, CD fetch)	IPA input to stage 2 (CD address)	Stage 2 fault on CD fetch
		`0b10` (S2, TT fetch)	IPA input to stage 2 (TTD address)	Stage 2 fault on stage 1 TTD fetch
		`0b11` (S2, IN)	IPA input to stage 2 (stage 1 output)	Stage 2 fault on IPA
	F_WALK_EABT	`0b00`	0	Stage 1 TTD fetch abort
		`0b01`, `0b10`, `0b11`	0	Stage 2 TTD fetch abort (for CD/TT/IN, as indicated by the REASON field)
	F_CD_FETCH	`0b00`	0	CD fetch abort
	MISC	`0b00`	0	

9.1.5 ATOS_PAR.FAULTCODE encodings

ATOS_PAR.FAULTCODE has the following values:

TYPE	Meaning	Notes
`0xFF`	INV_REQ	Malformed ATOS request
`0xFE`	INV_STAGE	Request for stage of translation that is not present
`0xFD`	INTERNAL_ERR	Translation was terminated for miscellaneous reason.
`0x02`	C_BAD_STREAMID	ATOS_SID.STREAMID out of range of configured Stream table. Note: ATOS_SID.STREAMID might not store bits above the

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

TYPE	Meaning	Notes
		implemented StreamID size. An implementation is permitted but not required to record this error if ATOS_SID.STREAMID is greater than the implemented StreamID size, or might truncate STREAMID to the implemented StreamID size.
0x03	F_STE_FETCH	External abort or error on STE fetch
0x04	C_BAD_STE	Selected STE invalid or illegal
0x06	F_STREAM_DISABLED	Non-substream request was made (ATOS_SID.SSID_VALID == 0) on configuration with Config[0] == 1 and STE.S1CDMax > 0 and STE.S1DSS == 0b00, or a request with SubstreamID 0 was made on configuration with Config[0] == 1 and STE.S1CDMax > 0 and STE.S1DSS == 0b10.
0x08	C_BAD_SUBSTREAMID	ATOS_SID.SSID out of range of configured CD table. Note: ATOS_SID.SUBSTREAMID might not store bits above the implemented SubstreamID size. An implementation is permitted but not required to record this error if ATOS_SID.SUBSTREAMID is greater than the implemented SubtreamID size, or might truncate SUBSTREAMID to the implemented SubstreamID size.
0x09	F_CD_FETCH	External abort or error on CD fetch For a VATOS (or other stage 1-only) request, this is returned where the CD fetch caused a stage 2 fault.
0x0A	C_BAD_CD	Selected CD invalid or illegal
0x0B	F_WALK_EABT	External abort or error on translation table walk For a VATOS (or other stage 1-only) request, this is returned where the stage 1 walk caused a stage 2 fault.
0x10	F_TRANSLATION	Translation fault
0x11	F_ADDR_SIZE	Address Size fault
0x12	F_ACCESS	Access flag fault
0x13	F_PERMISSION	Permission fault
0x20	F_TLB_CONFLICT	Translation caused TLB conflict condition
0x21	F_CFG_CONLICT	Translation caused configuration cache conflict condition

All other values are RESERVED.

The priority order of the C_* and F_* errors and faults follow the same order that is set out for Event queue events in 7.3.21, with the relative priorities for INV_REQ and INV_STAGE as shown her:

1. INV_REQ
    a. Note: Whether INV_REQ is to be reported is determined statically before access of configuration structures.
2. INV_STAGE (point A)

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

3. C_BAD_STREAMID
4. F_STE_FETCH
5. C_BAD_STE
6. INV_STAGE (point B)
7. C_BAD_SUBSTREAMID
8. F_STREAM_DISABLED
9. Translation faults (stage 2, for fetch of a CD)
10. F_CD_FETCH
11. C_BAD_CD
12. Translation-related faults (stage 1 or stage 2, for input address)

INV_STAGE is reported after the STE is fetched and valid, at point B, after C_BAD_STREAMID, F_STE_FETCH and C_BAD_STE are checked, if the enabled set of stages of the STE are determined to be incompatible with the request. INV_STAGE takes precedence over C_BAD_SUBSTREAMID, F_STREAM_DISABLED, F_CD_FETCH, C_BAD_CD and all Translation-related faults. INV_STAGE is permitted to be determined statically prior to checking for C_BAD_STREAMID at point A, in the case where a secure ATOS request is made with SMMU_S_GATOS_SID.SSEC == 1.

9.1.6 SMMU_VATOS_SEL

ARM expects the VATOS interface to be exposed to a single guest virtual machine at a time. The StreamID provided by the VM for translation using SMMU_VATOS_SID is verified by comparing it to the selected STE's S2VMID field against the SMMU_VATOS_SEL.VMID field. If the values match, the operation completes normally. If the values do not match, any request for information about that StreamID is denied with C_BAD_STE. The STE's S2VMID field is only valid, and can only match the SMMU_VATOS_SEL.VMID field, when the STE is associated with the NS-EL1 StreamWorld. If a VATOS request uses an STE associated with any other StreamWorld, the request results in C_BAD_STE because STE.S2VMID is not valid and therefore does not match. This applies even if the configuration does not translate at stage 2.

Note: ARM expects the VATOS interface to be used in scenarios where a guest is able to make queries using a physical StreamID number and where a system-specific mechanism describes the presence of the VATOS page in the guest address space.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

10 Performance Monitor Extension

10.1 Support and discovery

Performance monitoring facilities are optional. When implemented, the SMMU has one or more Performance Monitor Counter Groups (PMCG) associated with it. The discovery and enumeration of a base address for each group is IMPLEMENTATION DEFINED. The Performance Monitor Counter Groups are standalone monitoring facilities and, as such, can be implemented in separate components that are all associated with (but not necessarily part of) an SMMU.

Note: A centralised ID scheme to identify common properties is not currently exploited because PMCGs might be independently-designed, and contain their own identification facilities.

10.2 Overview of counters and groups

The SMMU performance monitoring facility provides:

- A number of specified events can be detected that correspond to behavior of aspects of the SMMU or the implementation.
- A number of counters that increment when their configured event occurs.

An implementation has one or more counter groups, and each group contains 1 - 64 counters. Each group has a discoverable set of supported events which can be counted by the counters within the group. All counters in a group can be configured to count any of the supported events. Different groups are not required to support the same set of events.

For example, a particular distributed SMMU implementation might have two groups,

- A group for a unit performing translation table walks and counting translation-related events
- A group for a separate unit containing TLBs and counting TLB-related events.

Some event types are not attributable to one traffic StreamID and are considered to be common to all. Other events are generated by a specific stream and can be filtered by StreamID so that a counter increment is enabled when the StreamID filter matches the event's originating StreamID.

Counters associated with a group are not required to be able to count events that originate from all possible StreamIDs. A counter group is permitted to be affiliated with a subset of incoming StreamIDs. ARM strongly recommends that the incoming StreamID namespace is covered by the union of all counter groups, so that events from any StreamID can be counted using a counter in a group appropriate to the StreamID value. The discovery of whether a given group counts events from a given StreamID, that is the mapping of a StreamID to supporting group or groups, is IMPLEMENTATION DEFINED.

Multiple counter groups might count events that originate from the same StreamID or StreamIDs, or from overlapping sets of StreamIDs.

Note: Because counter groups are not required to support the same kinds of events, events that originate from a given StreamID might be counted in several groups, if multiple groups are associated with that StreamID.

Note: Taking the previous example of the distributed implementation, activity from a given StreamID might cause TLB miss events that can be counted in the group associated with a TLB unit, whereas the same StreamID might

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

cause translation table fetch events that can only be counted in the other counter group associated with the translation table walk unit.

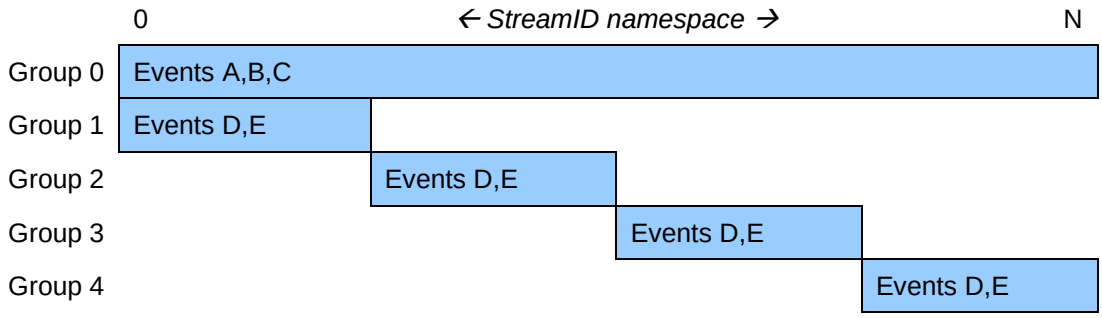


Figure 19: Example counter groups filtered on events from different StreamID spans

Figure 19 illustrates these rules. In this example system, five counter groups exist and any event A, B, C, D or E can be counted for any StreamID. For events A, B and C, any counter in Group 0 can count events that originate from any input StreamID. However, events D and E are counted by groups 1-4 and the group that is selected depends on which services the relevant StreamIDs. For example, to count event D originating from StreamID N, a counter in Group 4 must be used.

Note: Although a counter can be configured to count an event that is filtered by StreamID N, a counter can also be configured to count an event without the StreamID filter. In the example in Figure 19, a counter in Group 4 could count an event that occurs from any of the StreamIDs associated with Group 4.

If a counter group exists that is associated with all StreamIDs, ARM recommends that Group 0 is used.

The association between counter groups and ranges of StreamIDs is fixed at design time and no mechanism is provided to re-configure this mapping.

A counter group is conceptually free-standing and has no interdependency on the behavior of other counter groups or even on the SMMU configuration itself. This means that:

- A counter group can be dissimilar to other groups, for example if implemented in a different IP block to another group.
- The capabilities of a given counter group (events captured, number of counters, MSI support) might be unique in the SMMU

If the SMMUEN for a Security state is zero, it is IMPLEMENTATION DEFINED whether events that can be filtered by stream are counted for activity on the streams that are associated with that Security state. The Secure Observation rules in 10.6 also apply if the implementation supports counting the event in this scenario.

For events that cannot be filtered by stream, it is IMPLEMENTATION DEFINED whether they are counted when all traffic through the SMMU is treated as global bypass, as indicated by SMMU_CR0.SMMUEN == 0 and, if implemented, SMMU_S_CR0.SMMUEN == 0.

Note: Event 0, “Clock cycle” is the only architectural event in this category.

Unless otherwise specified, when the configuration is changed, there is an implementation-specific delay between the observation of the register write by the PMCG and the configuration change taking effect. The change is not required to affect transactions that have been observed and might be partly processed by the SMMU.

Note: ARM expects that any delay is small. Register accesses and client transactions that occur after the completion of a register access that alters the configuration are expected to observe the new configuration.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

10.2.1 Overflow, interrupts and capture

An individual counter overflows when an increment causes a carry-out to be generated, that is when the counter wraps beyond its maximum unsigned value to a smaller value. When this happens, a bit is set in the Overflow Status array that corresponds to the counter that overflowed. An overflow condition does not prevent a counter from counting.

Note: Software can read the Overflow Status (OVS) array using the SMMU_PMCG_OVS{SETn,CLRn} registers to determine which counters have overflowed.

A counter can be programmed to assert a common per-counter group interrupt, when enabled, on overflow. In addition, the counter group interrupt has a per-group overall enable flag.

Note: Whether an overflow occurs is independent of, and not prevented by, the OVS status. If a counter overflow occurs, any side effects of interrupting and capturing it, if enabled, occur regardless of the value of the corresponding OVS bit.

The counter group interrupt is an edge-triggered wired interrupt output, an MSI interrupt, or both. Because counter groups might represent distinct IP blocks, MSI support can differ between counter groups that are associated with one SMMU. However, ARM strongly recommends that all counter groups support MSIs if the core SMMU supports MSIs.

Note: As with the core SMMU MSI IRQs, an implementation that supports MSIs can still supply a wired IRQ output, giving the system the choice of using the wired IRQ and leaving the MSI disabled, or using the MSI.

Optionally, a PMCG can allow all counter registers in a group to be simultaneously sampled by a capture mechanism that copies the counter values to their respective shadow register. The presence of support for this mechanism is discovered via SMMU_PMCG_CFGR.CAPTURE == 1. The capture mechanism is triggered by one of the following:

- A write of 1 to SMMU_PMCG_CAPR.CAPTURE.
- Overflow of a counter configured with SMMU_PMCG_EVTYPERn.OVFCAP == 1.
- An external IMPLEMENTATION DEFINED mechanism.

When capture is triggered by a write to SMMU_PMCG_CAPR.CAPTURE, the captured values are observable to an access that is performed after the write to SMMU_PMCG_CAPR completes.

Note: One use model for the PMCG might be to program a counter to overflow within a known number of counts and, on overflow a capture is triggered and an interrupt is sent. The interrupt handler reads out the captured counter values.

Note: In addition, counter values and their overflow events can be exported in an IMPLEMENTATION DEFINED manner to external profiling logic. Export and capture of counter values through a mechanism that is invisible to the SMMU PMCG programming interface are unrelated to the presence of the programmatic capture mechanism indicated by SMMU_PMCG_CFGR.CAPTURE.

If an MSI is configured with a cacheable, shareable attribute, whether the PMCG is able to perform a cache-coherent MSI write is IMPLEMENTATION DEFINED. The core SMMU_IDR0.COHACC field does not apply to PMCG MSI writes.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When a counter that overflows is configured to trigger an interrupt, the interrupt is made observable after all of the following have become observable:

- The update to OVS.
- The new counter value.
- The captured values, if the overflow that caused the interrupt also triggered a capture.

10.3 Monitor events

Performance events are indicated by a numeric ID, in the following ranges:

- `0x0000` to `0x007F`: Architected events
- `0x0080` to `0xFFFF`: IMPLEMENTATION DEFINED events

The architecture defines the following event types and their causes:

Event ID	Description	Notes	Mandatory	Can filter on StreamID
0	Clock cycle	The period between these events is IMPLEMENTATION DEFINED. It is IMPLEMENTATION DEFINED whether this event is derived from a continuous time base or an SMMU clock which might be subject to implementation specific gating based on SMMU activity.	Y	N
1	Transaction	For a component accepting upstream client device transactions, a transaction passed through the component and issued into system, whether TLB/cache misses occurred or not. This event includes: • Ordinary transactions that bypass translation for any reason. • Ordinary transactions that have undergone translation. • PCIe ATS Translated transactions whether subject to further Stage 2 translation (EATS == `0b10`) or whether bypassing translation. This event does not include transactions terminated by the SMMU. For a component performing translation services, a translation request was received from another component; this includes ATS Translation Requests.	Y	Y

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Event ID	Description	Notes	Mandatory	Can filter on StreamID
2	TLB miss caused by incoming transaction or (ATS or non-ATS) translation request	As TLBs have missed, a translation operation was required using an internal translation table walker or from another component providing walk services. Where multiple levels of TLB are implemented in a component, this event indicates all TLBs missed (therefore a lookup was required).	Y	Y
3	Configuration cache miss caused by transaction or (ATS or non-ATS) translation request	As configuration cache has missed, a configuration structure fetch was required using an internal structure walker or from another component providing walk services. Where multiple levels of configuration cache are implemented in a component, this event indicates all caches missed (therefore a lookup was required).	Y	Y
4	Translation table walk access	External transactions performed by a translation table walker, including accesses performed speculatively. This applies to translation table walks performed for any reason (transactions, non-ATS translation requests, ATS Translation Requests, ATOS, HTTU or prefetch operations).	Y	Y
5	Configuration structure access	External transactions performed by a configuration table walker, including accesses performed speculatively. This applies to configuration fetches performed for any reason.	Y	Y
6	PCIe ATS Translation Request received	Independent of success/failure of response. Note: ATS translation failures are visible at the endpoint, either through an error or because of the endpoint causing a software-visible PRI Page Request.	If ATS supported	Y
7	PCIe ATS Translated Transaction passed through SMMU	Transaction completed translation and issued into system, whether TLB/cache misses occurred or not. This event includes both ATS Translated transactions subject to further stage 2 translation (EATS == `0b10`) and those that bypass translation. Note: Translated transactions might bypass or be subject to further translation, as STE.EATS dictates.	If ATS supported	Y

When an event is mandatory, the following are both true:

- The event is supported in at least one counter group.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- The event is supported in as many groups as required to make the event countable for all possible StreamIDs in use.
- Note: In a component that only caches TLB and configuration data in a single combined cache, events 2 and 3 count the same occurrence (a miss of the combined cache is a miss of both configuration and translation).

Note: As an example, a distributed implementation might have different roles for different components (and therefore different PMCGs), so that one component might translate transactions from client devices and another component handles translation requests for the former. The first component has no table walker so it is not required to count events 4 and 5. These events are instead counted by the second component, which performs configuration and page table walks. The second component could still include TLBs and configuration caches, in which case it would also count events 2 and 3.

10.4 StreamIDs and filtering

Some event types might be filtered by StreamID, so that events are only counted if they are associated with transactions that match the configured StreamID filter. Other event types are considered unrelated to any particular StreamID and are counted by any counter that is configured to count the event (any StreamID filtering configuration is ignored). An implementation provides a StreamID filter per counter, or one StreamID filter that applies to all counters in the PMCG. This is identified through the SMMU_PMCG_CFGR.SID_FILTER_TYPE field.

When SMMU_PMCG_CFGR.SID_FILTER_TYPE == 0, the StreamID filter configuration is controlled by register fields for each counter. SMMU_PMCG_EVTYPEn.{FILTER_SID_SPAN,FILTER_SEC_SID} and SMMU_PMCG_SMRn.STREAMID are associated with counter n as accessed through SMMU_PMCG_EVCNTRn.

When SMMU_PMCG_CFGR.SID_FILTER_TYPE == 1, the fields SMMU_PMCG_EVTYPE0.{FILTER_SID_SPAN,FILTER_SEC_SID} and SMMU_PMCG_SMR0.STREAMID apply to all counters in the group.

When the corresponding counter is configured to count an event type that can be filtered by StreamID, this configuration allows filtering by:

- Exact StreamID match.
- Partial StreamID match where a variable span of StreamID[N:0] are ignored and only upper StreamID bits are required to match the given filter.
- Match any StreamID (filtering disabled).

When SMMU_PMCG_EVTYPEn.FILTER_SID_SPAN == 0, SMMU_PMCG_SMRn.STREAMID is programmed with a StreamID that is matched exactly. Events that are associated with other StreamIDs do not cause an increment of the counter.

When SMMU_PMCG_EVTYPEn.FILTER_SID_SPAN == 1:

- To match a partial StreamID where the X most-significant bits must match but the ‘Y’ least-significant bits might differ, SMMU_PMCG_SMRn.STREAMID is programmed with a value that contains:
    - STREAMID[Y - 1] == 0.
    - STREAMID[Y - 2:0] == 1 (where Y > 1).
    - The remainder of implemented bits of STREAMID (X bits, from bit Y upwards) contain a value to match from the corresponding bits of event StreamID.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Examples (in binary):
  - `0000:0000:0001:1011:1111:0111:1111:0111` matches `0000:0000:0001:1011:1111:0111:1111:xxxx`
  - `0000:0000:0001:1011:1111:0111:1111:0110` matches `0000:0000:0001:1011:1111:0111:1111:011x`
  - `0000:0000:0001:1011:1111:0101:1111:1111` matches `0000:0000:0001:1011:1111:01xx:xxxx:xxxx`

- To match any StreamID, SMMU_PMCG_SMRn.STREAMID is programmed with 1 for all implemented bits:
  - Note: The STREAMID field might implement fewer bits than the SMMU StreamID size. ARM recommends that `0xFFFFFFFF` is written to this field to ensure that all implemented bits are set without first determining how many bits to write.
  - Note: A value with 0 in the most-significant implemented bit and 1 in all bits below that point is an alternative encoding that matches any StreamID. This is the maximum mask given the encoding of the variable mask size, but requires knowledge of the implemented field size. When security is supported, this encoding behaves differently to the 'all 1' encoding in terms of the scope of the security.

When the PMCG implements support for two Security states (see section 10.6):

- The SMMU_PMCG_EVTYPERn.FILTER_SEC_SID flag determines whether the StreamID filter configuration (as determined by SMMU_PMCG_EVTYPERn.FILTER_SID_SPAN and SMMU_PMCG_SMRn.STREAMID) applies to the Secure or the Non-secure StreamID namespace. Secure observation can be disabled so that StreamID matching can only match events that are associated with Non-secure StreamIDs, overriding the effective value of this flag.
- When SMMU_PMCG_EVTYPERn.FILTER_SID_SPAN == 1 and SMMU_PMCG_SMRn.STREAMID field is programmed with 1 in all implemented bits, the filter matches all StreamIDs for both Secure and Non-secure namespaces if Secure observation is enabled, and matches all non-secure StreamIDs if Secure observation is not enabled. In SMMUv3.0, it is IMPLEMENTATION DEFINED whether:
  - The filter matches all StreamIDs in one of the Secure or Non-secure namespaces as determined by the effective value of the SMMU_PMCG_EVTYPERn.FILTER_SEC_SID flag.
  - The filter matches all StreamIDs for both Secure and Non-secure namespaces if Secure observation is enabled, and matches all Non-secure StreamIDs if Secure observation is not enabled.
- When SMMU_PMCG_EVTYPERn.FILTER_SID_SPAN == 1 and SMMU_PMCG_SMRn.STREAMID is programmed with 0 in the most-significant implemented bit and 1 in all other implemented bits, the filter matches all StreamIDs for the Secure or the Non-secure namespace as determined by the effective value of the SMMU_PMCG_EVTYPERn.FILTER_SEC_SID flag.

10.4.1 Counter Group StreamID size

The StreamID that is handled by a PMCG is not required to be the full SMMU-architectural StreamID as seen by the SMMU programming interface. This arises from the possibility that a PMCG represents a sub-component of the SMMU and, in a distributed implementation, the component might only service a subset of the StreamID space of the SMMU. In such an implementation, the upper bits of StreamID might be considered fixed for a given sub-component. For example, block 0 serves clients with StreamIDs `0x00` to `0x0F` and block 1 serves clients with StreamIDs `0x10` to `0x1F`.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

In all cases, the low-order PMCG StreamID bits [N:0] must be equivalent to the SMMU StreamID[N:0]. Where a PMCG StreamID filter is programmed, SMMU_PMCG_SMRn.STREAMID might implement fewer bits than indicated by SMMU_IDR1.SIDSIZE in the SMMU with which the PMCG is associated. The implemented size of the SMMU_PMCG_SMRn.STREAMID field is IMPLEMENTATION DEFINED. The association between the span of StreamIDs served by a given PMCG and the overall SMMU StreamID namespace is IMPLEMENTATION DEFINED.

For example, an SMMU with a 17-bit StreamID might be composed of two components A and B, which support client devices with StreamIDs `0` to `0xFFFF` and `0x10000` to `0x1FFFF` respectively. In order to count events from the client device with StreamID 0x12345, software programs a counter in the PMCG that is associated with component B to filter on StreamID `0x12345`. However, SMMU_PMCG_SMRn.STREAMID might only implement 16 bits and read back the value `0x02345`.

Software must not depend on readback of SMMU_PMCG_SMRn.STREAMID returning the full SMMU StreamID. The readback value might be truncated to the PMCG StreamID size.

10.5 Registers

The total number of counter groups that are associated with one SMMU is IMPLEMENTATION DEFINED. Each counter group is represented by one 4KB page (Page 0) with one optional additional 4KB page (Page 1), both of which are at IMPLEMENTATION DEFINED base addresses.

If present, Page 1 contains alternate locations for counter values, overflow status and shadow capture control. Presence is indicated by SMMU_PMCG_CFGR.RELOC_CTRS. ARM recommends that Page 1 is implemented and, if so, ARM strongly recommends that Page 1 is located within an aligned 64KB system address span that is otherwise unused.

Note: This permits a hypervisor to use a 64KB stage 2 granule to expose the Page 1 counter values for direct access by a virtual machine. ARM expects that guest access to stage 1 registers (for counter configuration) will be trapped and controlled by the hypervisor, rather than accessed directly.

Access behaviors follow the same rules as for the SMMU registers described in section 6.2. In particular:

- Aligned 32-bit access is permitted to 64-bit registers.
- It is IMPLEMENTATION DEFINED whether 64-bit accesses to 64-bit registers are atomic.
- All registers are little-endian.

Unless otherwise specified, software is permitted to read or write a register at any time. Writes to read-only registers are IGNORED.

10.5.1 SMMU_PMCGn address map

The map of the base PMCG register Page 0 is shown here:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Page 0 offset	Name	Access	Notes
`0x000 + (n* stride)` (`0x000` to `0x1FF`)	SMMU_PMCG_EVCNTRn	32 or 64 RW	Counter value, 0 ≤n < 64. Registers placed on a 4-byte stride if SMMU_PMCG_CFGR.SIZE ≤31 (counters 32-bit or smaller), otherwise an 8-byte stride.
`0x400 + (n*4)` (`0x400` to `0x4FF`)	SMMU_PMCG_EVTYPERn	32 RW	Counter event type configuration, 0 ≤n < 64.
`0x600 + (n* stride)` (`0x600` to `0x7FF`)	SMMU_PMCG_SVRn	32 or 64 RO	Shadow value, 0 ≤n < 64. Stride same as for SMMU_PMCG_EVCNTRn.
`0xA00 + (n*4)` (`0xA00` to `0xAFF`)	SMMU_PMCG_SMRn	32 RW	Counter stream match filter, 0 ≤n < 64.
`0xC00`	SMMU_PMCG_CNTENSET0	64 RW	Counter enable SET.
`0xC20`	SMMU_PMCG_CNTENCLR0	64 RW	Counter enable CLEAR.
`0xC40`	SMMU_PMCG_INTENSET0	64 RW	Counter interrupt contribution enable SET.
`0xC60`	SMMU_PMCG_INTENCLR0	64 RW	Counter interrupt contribution enable CLEAR.
`0xC80`	SMMU_PMCG_OVSCLR0	64 RW	Overflow status CLEAR.
`0xCC0`	SMMU_PMCG_OVSSET0	64 RW	Overflow status SET.
`0xD88`	SMMU_PMCG_CAPR	32 WO	Counter shadow value capture
`0xDF8`	SMMU_PMCG_SCR	32 RW Secure	Secure Control Register
`0xE00`	SMMU_PMCG_CFGR	32 RO	PMCG configuration information
`0xE04`	SMMU_PMCG_CR	32 RW	Control Register
`0xE20`	SMMU_PMCG_CEID0	64 RO	Common Event ID bitmap, lower.
`0XE28`	SMMU_PMCG_CEID1	64 RO	Common Event ID bitmap, upper.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Page 0 offset	Name	Access	Notes
0xE50	SMMU_PMCG_IRQ_CTRL	32 RW	PMCG IRQ enable
0xE54	SMMU_PMCG_IRQ_CTRLACK	32 RO	PMCG IRQ enable acknowledge
0xE58	SMMU_PMCG_IRQ_CFG0	64 RW	PMCG MSI configuration
0xE60	SMMU_PMCG_IRQ_CFG1	32 RW	PMCG MSI configuration
0xE64	SMMU_PMCG_IRQ_CFG2	32 RW	PMCG MSI configuration
0xE68	SMMU_PMCG_IRQ_STATUS	32 RO	PMCG MSI status Note: This location is RES0 in SMMUv3.0.
0xE70	SMMU_PMCG_AIDR	32 RO	Architecture Identification
0xE80 to 0xEFF	IMPLEMENTATION DEFINED		
0xFB0 to 0xFFF	SMMU_PMCG_ID_REGS		IMPLEMENTATION DEFINED

When Page 1 is present (SMMU_PMCG_CFGR.RELOC_CTRS == 1), the following registers are relocated to Page 1 and their corresponding Page 0 locations become RES0:

Page 1 offset	Name	Access	Notes
0x000 + (n* stride) (0x000 to 0x1FF)	SMMU_PMCG_EVCNTRn	Same as for corresponding locations in Page 0.	
0x600 + (n* stride) (0x600 to 0x7FF)	SMMU_PMCG_SVRn		
0xC80	SMMU_PMCG_OVSCLR0		
0xCC0	SMMU_PMCG_OVSSET0		
0xD88	SMMU_PMCG_CAPR		

10.5.2 Register details

10.5.2.1 SMMU_PMCG_EVCNTRn

Read-write.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When counter size is ≤32 bits (SMMU_PMCG_CFGR.SIZE ≤31), these registers are 32 bits in size. Otherwise, these registers are 64 bits in size. Present in an array of n registers, all of size 32 or 64, each corresponding to counter n.

- [N:0] Counter Value

  R == SMMU_PMCG_CFGR.SIZE + 1

  If R < 32, bits [31:R] are RES0.

  If R > 32 and R < 64, bits [63:R] are RES0.

Resets to an UNKNOWN value.

The counter value is incremented when an event matching SMMU_PMCG_EVTYPERn.EVENT occurs, the counter is enabled (the respective CNTEN == 1) and, if the event type is filterable by StreamID, filters match. See section 10.4 for information on filtering.

When a counter value is incremented by the PMCG, the increment is atomic with respect to external writes to this register.

Registers corresponding to unimplemented counters are RES0.

10.5.2.2 SMMU_PMCG_EVTYPERn

Read-write.

- [31] OVFCAP
  - `0b0`: An overflow of counter n does not trigger a capture of all counters.
  - `0b1`: An overflow of counter n triggers a capture of all counters in the same way as by SMMU_PMCG_CAPR.CAPTURE.
  - When SMMU_PMCG_CFGR.CAPTURE == 0, this bit is RES0.

- [30] FILTER_SEC_SID
  - `0b0`: Count events originating from Non-secure StreamIDs.
  - `0b1`: Count events originating from Secure StreamIDs.
  - This bit is RES0 if the PMCG does not implement Security support. Otherwise, this bit is effectively 0 if Security support is implemented but Secure observation is disabled (SMMU_PMCG_SCR.SO == 0).
  - For event types that can be filtered on StreamID, this bit causes the StreamID match determined by FILTER_SID_SPAN and SMMU_PMCG_SMRn.STREAMID to match Secure or Non-secure StreamIDs (as determined by SMMU SSD). See section 10.4.
  - Where the StreamID match span encodes ALL, this bit selects counting of events associated with all Non-secure StreamIDs or all Secure StreamIDs.

- [29] FILTER_SID_SPAN
  - `0b0`: SMMU_PMCG_SMRn.STREAMID filters event on an exact StreamID match, if the event type can be filtered on StreamID.
  - `0b1`: The SMMU_PMCG_SMRn.STREAMID field encodes a match span of StreamID values. See 10.4.
    - Note: The span can encode ALL, equivalent to disabling filtering on StreamID.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [28:16] Reserved, RES0.

- [15:0] EVENT
  - Event type that causes this counter to increment.
  - An IMPLEMENTATION DEFINED number of low-order bits of this register are implemented, unimplemented upper bits are RES0.

Resets to an UNKNOWN value.

These registers contain per-counter configuration, in particular the event type counted.

When SMMU_PMCG_CFGR.SID_FILTER_TYPE == 0, the field FILTER_SID_SPAN and, if Security is supported, FILTER_SEC_SID, are present in every implemented SMMU_PMCG_EVTYPERn register and correspond to counter SMMU_PMCG_EVCNTRn for all implemented values of n.

When SMMU_PMCG_CFGR.SID_FILTER_TYPE == 1, these fields are present only in SMMU_PMCG_EVTYPER0 and the FILTER_SID_SPAN and FILTER_SEC_SID fields in SMMU_PMCG_EVTYPERx, for x ≥1, are RES0.

See SMMU_PMCG_SMRn and 10.4 for information on StreamID filtering. When filtering is enabled, an event is counted only if it matches the filter conditions for the counter.

Note: Event types that cannot be filtered on StreamID are always counted, as they cannot be filtered out using FILTER_* configuration.

Registers corresponding to unimplemented counters are RES0.

10.5.2.3 SMMU_PMCG_SVRn

Read-only.

When counter size is ≤32 bits (SMMU_PMCG_CFGR.SIZE ≤31), these registers are 32 bits in size. Otherwise, these registers are 64 bits in size. Present in an array of registers (all of size 32 or 64) each corresponding to counter n.

- [N:0] Shadow Counter Value

  R == SMMU_PMCG_CFGR.SIZE + 1

  If R < 32, bits [31:R-1] are RES0.

  If R > 32 and R < 64, bits [63:R-1] are RES0.

Resets to an UNKNOWN value.

When counter value capture is implemented (SMMU_PMCG_CFGR.CAPTURE == 1), these registers hold the captured counter values of the corresponding entries in SMMU_PMCG_EVCNTRn. Registers corresponding to unimplemented counters are RES0.

If counter value capture is not implemented (SMMU_PMCG_CFGR.CAPTURE == 0), all SMMU_PMCG_SVRn registers are RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

10.5.2.4 SMMU_PMCG_SMRn

Read-write.

- [31:0] STREAMID
  - When the corresponding SMMU_PMCG_EVTYPERn.EVENT indicates an event that cannot be filtered on StreamID, the value in this register is IGNORED.
  - Otherwise:
    - When the corresponding SMMU_PMCG_EVTYPERn.FILTER_ID_SPAN == 0, the respective counter only counts events associated with a StreamID matching this field exactly.
    - When FILTER_ID_SPAN == 1, this field encodes a mask value that allows a span of least-significant StreamID bits to be ignored for the purposes of filtering on a StreamID match. When all implemented bits of this field are written to 1, any StreamID is matched. See section 10.4.
    - When Secure observation is enabled, SMMU_PMCG_EVTYPERn.FILTER_SEC_SID determines whether the StreamID is matched from Secure or Non-secure StreamID spaces (as defined by SMMU SSD), see section 10.4 for the behavior of the match-all encodings with respect to Secure/Non-secure namespaces.
  - This field implements an IMPLEMENTATION DEFINED number of contiguous bits (from 0 upwards) corresponding to the PMCG StreamID size, see section 10.4.1. Bits outside this range read as zero, writes ignored (RAZ/WI).

Resets to an UNKNOWN value.

These registers contain StreamID-match configuration for filtering events on StreamID.

When SMMU_PMCG_CFGR.SID_FILTER_TYPE == 0, each SMMU_PMCG_SMRn corresponds to counter SMMU_PMCG_EVCNTRn for all implemented values of n.

When SMMU_PMCG_CFGR.SID_FILTER_TYPE == 1, SMMU_PMCG_SMR0 applies to all counters in the group and registers SMMU_PMCG_SMRx for x≥1 are RES0.

See section 10.4 for more information on StreamIDs in PMCGs.

Note: To count events for a given StreamID, software must choose a counter in the appropriate counter group to use for the required event type (which is determined in an IMPLEMENTATION DEFINED manner) and then write the full StreamID value into the STREAMID field of this register.

Registers corresponding to unimplemented counters are RES0.

10.5.2.5 SMMU_PMCG_CNTENSET0

Read-write.

- [63:0] CNTEN

Resets to an UNKNOWN value.

64-bit register containing a bitmap of per-counter enables. Bit CNTEN[n] corresponds to counter n, which is enabled if CNTEN[n] == 1 and SMMU_PMCG_CR.E == 1.

Write 1 to a bit location to set the per-counter enable. Reads return the state of enables. High-order bits of the bitmap beyond the number of implemented counters are RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

10.5.2.6 SMMU_PMCG_CNTENCLR0

Read-write.

- [63:0] CNTEN

Resets to an UNKNOWN value.

Bitmap indexed similar to SMMU_PMCG_CNTENSET0.

Write 1 to a bit location to clear the per-counter enable. Reads return the state of enables. High-order bits of the bitmap beyond the number of implemented counters are RES0.

10.5.2.7 SMMU_PMCG_INTENSET0

Read-write.

- [63:0] INTEN

Resets to an UNKNOWN value.

Bitmap indexed similar to SMMU_PMCG_CNTENSET0.

Write 1 to a bit location to set the per-counter interrupt enable. Reads return the state of interrupt enables. High-order bits of the bitmap beyond the number of implemented counters are RES0.

Overflow of counter n triggers a PMCG interrupt if, at the time that the overflow occurs, the corresponding INTEN[n] == 1 && SMMU_PMCG_IRQ_CTRL.IRQEN == 1.

10.5.2.8 SMMU_PMCG_INTENCLR0

Read-write.

- [63:0] INTEN

Resets to an UNKNOWN value.

Bitmap indexed similar to SMMU_PMCG_CNTENSET0.

Write 1 to a bit location to clear the per-counter interrupt enable. Reads return the state of interrupt enables. High-order bits of the bitmap beyond the number of implemented counters are RES0.

10.5.2.9 SMMU_PMCG_OVSCLR0

Read-write.

- [63:0] OVS

Resets to an UNKNOWN value.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bitmap indexed similar to SMMU_PMCG_CNTENSET0.

Write 1 to a bit location to clear the per-counter overflow status. Reads return the state of overflow status. High-order bits of the bitmap beyond the number of implemented counters are RES0.

Overflow of counter n (a transition past the maximum unsigned value of the counter causing the value to become, or pass, zero) sets the corresponding OVS[n] bit. In addition, this event can trigger the PMCG interrupt and cause a capture of the PMCG counter values (see SMMU_PMCG_EVTYPERn).

10.5.2.10 SMMU_PMCG_OVSSET0

Read-write.

- [63:0] OVS

Resets to an UNKNOWN value.

Bitmap indexed similar to SMMU_PMCG_CNTENSET0.

Write 1 to a bit location to set the per-counter overflow status. Reads return the state of overflow status. High-order bits of the bitmap beyond the number of implemented counters are RES0.

Software setting an OVS bit is similar to an OVS bit becoming set because of a counter overflow, except it is implementation specific whether enabled overflow side-effects of triggering the PMCG interrupt or causing a capture of the PMCG counter values are performed.

10.5.2.11 SMMU_PMCG_CAPR

Write-only.

- [31:1] Reserved, RES0.
- [0] CAPTURE
  - When counter capture is supported (SMMU_PMCG_CFGR.CAPTURE == 1), a write of 1 to this bit triggers a capture of all SMMU_PMCG_EVCNTRn values within the PMCG into their respective SMMU_PMCG_SVRn shadow register.
  - When SMMU_PMCG_CFGR.CAPTURE == 0, this field is RES0.

This register reads as zero.

10.5.2.12 SMMU_PMCG_SCR

Read-write, Secure access only. Non-secure access has RAZ/WI behavior.

- [31] Reads as one
  - Secure software can use this bit to discover Security support in the PMCG.
- [30:3] Reserved, RES0.
- [2] NSMSI

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- `0b0`: Generated MSIs have an NS == 0 attribute
- `0b1`: Generated MSIs have an NS == 1 attribute
- This bit is RES0 if SMMU_PMCG_CFGR.MSI == 0. Otherwise, this bit resets to 1.

- [1] NSRA
  - `0b0`: Non-secure Register Access is disabled.
    - Non-secure access to any PMCG register is RAZ/WI.
  - `0b1`: Non-secure Register Access is enabled.
    - If the PMCG supports MSIs, generated MSIs have an NS == 1 attribute.
  - Resets to 1.
- [0] SO
  - `0b0`: Secure observation is disabled
    - SMMU_PMCG_EVTYPERn.FILTER_SEC_SID is effectively 0.
  - `0b1`: Secure observation is enabled
  - See section 10.6.
  - Resets to 0.

If the PMCG does not implement support for two Security states, this register is RAZ/WI.

If the PMCG does not implement two Security states but the system does, both Secure and Non-secure accesses are permitted to PMCG registers. In this case, any MSIs generated from the PMCG must be issued as NS == 1.

Note: If software transitions control of a PMCG between Non-secure and Secure realms, ARM recommends following this procedure:

- Set NSRA == 0 and NSMSI == 1
  - Non-secure register access is disabled, but any active MSI configuration remains Non-secure
- Clear SMMU_PMCG_IRQ_CTRL.IRQEN and wait for Update of SMMU_PMCG_IRQ_CTRLACK.
  - Outstanding Non-secure MSIs are complete.
- Set NSMSI == 0 and reprogram MSI registers as required.

10.5.2.13 SMMU_PMCG_CFGR

Read-only.

- [31:24] Reserved, RES0.

- [23] SID_FILTER_TYPE
  - `0b0`: Separate StreamID filtering is supported for each counter in the PMCG.
  - `0b1` The StreamID filter configured by SMMU_PMCG_SMR0 and SMMU_PMCG_EVTYPER0.{FILTER_SID_SPAN,FILTER_SEC_SID} applies to all counters in the PMCG.

- [22] CAPTURE
  - `0b0`: Capture of counter values into SVRn registers not supported.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

        - SMMU_PMCG_SVRn and SMMU_PMCG_CAPR are RES0.
    - `0b1`: Capture of counter values supported.

- [21] MSI Counter group supports Message Signalled Interrupt.
    - `0b0`: Group does not support MSI.
    - `0b1`: Group can send MSI.

- [20] RELOC_CTRS
    - When 1, Page 1 is present and the following registers are relocated to the equivalent offset on Page 1 (their Page 0 locations become RES0):
        - SMMU_PMCG_EVCNTRn.
        - SMMU_PMCG_SVRn.
        - SMMU_PMCG_OVSCLRn.
        - SMMU_PMCG_OVSSETn.
        - SMMU_PMCG_CAPR.

- [19:14] Reserved, RES0.

- [13:8] SIZE
    - Size of PMCG counters in bits, minus one. Valid values are:
        - 31 (32-bit counters).
        - 35 (36-bit counters).
        - 39 (40-bit counters).
        - 43 (44-bit counters).
        - 47 (48-bit counters).
        - 63 (64-bit counters).
    - Other values are Reserved.

- [7:6] Reserved, RES0.

- [5:0] NCTR
    - The number of counters available in the group is given by `NCTR+1`

The capability of each counter group to send MSIs is specific to the group and is not affected by the core SMMU MSI capability (SMMU_IDR0.MSI).

10.5.2.14 SMMU_PMCG_CR

Read-write.

- [31:1] Reserved, RES0.

- [0] E
    - Global counter enable, where 1 == Enabled. When 0, no events are counted and values in SMMU_PMCG_EVCNTRn registers do not change. This bit takes precedence over the CNTEN

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

bits set through SMMU_PMCG_CNTENSET0.

Resets to zero.

10.5.2.15 SMMU_PMCG_CEID{0,1}

Read-only.

128-bit bitmap comprised of two consecutive 64-bit registers. Bit (N & 63) of the 64-bit word at offset 8*(N/64) relates to event number N. For each bit,

- `0b0`: Event N cannot be counted by counters in this group.
- `0b1`: Event N can be counted by counters in this group.

See section 10.3 for event numbers.

10.5.2.16 SMMU_PMCG_IRQ_CTRL

Read-write.

- [31:1] Reserved, RES0.
- [0] IRQEN

Resets to zero.

Each field in this register has a corresponding field in SMMU_PMCG_IRQ_CTRLACK, with the same Update semantic as fields in SMMU_CR0 versus SMMU_CR0ACK.

This register contains the main IRQ enable flag for a per-counter group interrupt source. This enable allows or inhibits both edge-triggered wired outputs (if implemented) and MSI writes (if supported by the counter group). When IRQEN == 0, no interrupt is triggered regardless of individual per-counter overflow INTEN flags (that is, they are overridden). IRQEN also controls overall interrupt completion and MSI configuration changes.

When MSIs are supported, as indicated by SMMU_PMCG_CFGR.MSI == 1, IRQ enable flags Guard the MSI address and payload registers, which must only be changed when their respective enable flag is 0. See SMMU_PMCG_IRQ_CFG0 for details.

Completion of Update to IRQ enables guarantees the following side-effects:

- Completion of an Update of IRQEN from 0 to 1 guarantees that the MSI configuration in SMMU_PMCG_IRQ_CFG{0,1,2} will be used for all future MSIs generated from the counter group.
- An Update of IRQEN from 1 to 0 completes when all prior MSIs have become visible to their Shareability domain (have completed). Completion of this Update guarantees that no new MSI writes or wired edge events will later become visible from this source.

An IRQ is triggered from the PMCG if all of the following occur:

- SMMU_PMCG_IRQ_CTRL.IRQEN == 1
- A counter overflows and sets an OVS bit, whose corresponding INTEN bit was set through SMMU_PMCG_INTENSET0

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

10.5.2.17 SMMU_PMCG_IRQ_CTRLACK

Read-only.

- Same fields as SMMU_PMCG_IRQ_CTRL.

Resets to zero.

Undefined bits read as zero. Fields in this register are RES0 if their corresponding SMMU_PMCG_IRQ_CTRL field is Reserved.

10.5.2.18 SMMU_PMCG_IRQ_CFG0

Read-write.

- [63:52] Reserved, RES0.

- [51:2] ADDR[51:2] Physical address of MSI target register.
  - If ADDR == 0, no MSI is sent. This allows a wired IRQ, if implemented, to be used instead of an MSI.
  - Address bits above and below this field are treated as zero.
  - An IMPLEMENTATION DEFINED number of contiguous low-order bits of ADDR are implemented so that an address of at least the system PA size (see SMMU_IDR5.OAS) can be represented. The high-order bits of ADDR above the implemented span are RES0.
  - In SMMUv3.0, bits [51:48] are RES0.

- [1:0] Reserved, RES0.

This register resets to an UNKNOWN value.

When an implementation does not support MSIs, all fields are RES0.

SMMU_PMCG_IRQ_CFG0 is Guarded by SMMU_PMCG_IRQ_CTRL.IRQEN and must only be modified when IRQEN == 0. SMMU_PMCG_IRQ_CFG{0,1,2} have the same behaviors as SMMU_*_IRQ_CFG{0,1,2} with respect to their enables, including the permitted behaviors of writes when IRQEN == 1.

See SMMU_PMCG_SCR.NSMSI for control of the NS attribute of the MSI.

10.5.2.19 SMMU_PMCG_IRQ_CFG1

Read-write.

- [31:0] DATA MSI data payload.

This register resets to an UNKNOWN value.

When an implementation does not support MSIs, all fields are RES0.

SMMU_PMCG_IRQ_CFG1 is Guarded by SMMU_PMCG_IRQ_CTRL.IRQEN and must only be modified when IRQEN == 0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

10.5.2.20 SMMU_PMCG_IRQ_CFG2

Read-write.

- [31:6] Reserved, RES0.

- [5:4] SH[1:0] Shareability,
  - `0b00`: Non-shareable.
  - `0b01`: Reserved, behaves as `0b00`.
  - `0b10`: Outer Shareable.
  - `0b11`: Inner Shareable.
    - When MemAttr encodes a Device memory type, the value of this field is IGNORED and the Shareability is effectively Outer Shareable.

- [3:0] MemAttr Memory type.
  - Encoded the same as the STE.MemAttr field.

This register resets to an UNKNOWN value.

When an implementation does not support MSIs, all fields are RES0.

SMMU_PMCG_IRQ_CFG2 is Guarded by SMMU_PMCG_IRQ_CTRL.IRQEN and must only be modified when IRQEN == 0.

10.5.2.21 SMMU_PMCG_IRQ_STATUS

Read-only.

- [31:1] Reserved, RES0.

- [0] IRQ_ABT
  - The SMMU sets this bit to 1 if it detects that an MSI has terminated with an abort.
  - This bit is RES0 when SMMU_PMCG_CFGR.MSI == 0.
  - It is IMPLEMENTATION DEFINED whether an implementation can detect this condition.
  - This bit is cleared to 0 when SMMU_PMCG_IRQ_CTRL.IRQEN is Updated from 0 to 1.
    - Note: An IRQEN transition from 1 to 0 does not clear this bit, as this transition also ensures visibility of outstanding MSI writes and clearing IRQ_ABT at this point might mask possible abort completions of those MSI writes.

In SMMUv3.0, this register location is RES0.

10.5.2.22 SMMU_PMCG_AIDR

Read-only.

- [31:8] Reserved, RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- [7:4] ArchMajorRev
- [3:0] ArchMinorRev
    - [7:0] == `0x00` == SMMUv3.0 PMCG.
    - [7:0] == `0x01` == SMMUv3.1 PMCG.
    - All other values Reserved.

This register identifies the SMMU architecture version to which the PMCG implementation conforms.

10.5.2.23 SMMU_PMCG_ID_REGS

SMMU_PMCG register offsets `0xFB0-0xFFC` are defined as a read-only identification register space. For ARM implementations of the SMMU architecture the assignment of this register space, and naming of registers in this space, is consistent with the ARM identification scheme for CoreLink and CoreSight components. ARM strongly recommends that other implementers also use this scheme to provide a consistent software discovery model.

For ARM implementations, the following assignment of fields, consistent with CoreSight ID registers [8], is used:

Offset	Name (SMMU_PMCG_-prefixed)	Field	Value	Meaning
`0xFF0`	CIDR0, Component ID0	[7:0]	`0x0D`	Preamble
`0xFF4`	CIDR1, Component ID1	[7:4]	`0x9`	CLASS
		[3:0]	`0x0`	Preamble
`0xFF8`	CIDR2, Component ID2	[7:0]	`0x05`	Preamble
`0xFFC`	CIDR3, Component ID3	[7:0]	`0xB1`	Preamble
`0xFE0`	PIDR0, Peripheral ID0	[7:0]	IMPDEF	PART_0: bits [7:0] of the Part number
`0xFE4`	PIDR1, Peripheral ID1	[7:4]	IMPDEF	DES_0: bits [3:0] of the JEP106 Designer code
		[3:0]	IMPDEF	PART_1: bits [11:8] of the Part number
`0xFE8`	PIDR2, Peripheral ID2	[7:4]	IMPDEF	REVISION
		[3]	1	JEDEC-assigned value for DES always used
		[2:0]	IMPDEF	DES_1: bits [6:4] bits of the JEP106 Designer code
`0xFEC`	PIDR3, Peripheral ID3	[7:4]	IMPDEF	REVAND
		[3:0]	IMPDEF	CMOD
`0xFD0`	PIDR4, Peripheral ID4	[7:4]	0	SIZE
		[3:0]	IMPDEF	DES_2: JEP106 Designer continuation code
`0xFD4`	PIDR5, Peripheral ID5		RES0	Reserved
`0xFD8`	PIDR6, Peripheral ID6		RES0	Reserved

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Offset	Name (SMMU_PMCG_-prefixed)	Field	Value	Meaning
0xFDC	PIDR7, Peripheral ID7		RES0	Reserved
0xFB8	PMAUTHSTATUS	[7:0]	—	See below
0xFBC	PMDEVARCH	[31:21]	0x23B	ARCHITECT (0x23B is ARM's JEP106 code)
		[20]	1	PRESENT
		[19:16]	0	REVISION
		[15:0]	0x2A56	ARCHID
0xFCC	PMDEVTYPE	[7:4]	5	Sub-type: Associated with an SMMU
		[3:0]	6	Class: Performance monitor device type

Fields outside of those defined in this table are RES0.

Note: The Designer code fields (DES_*) fields for ARM-designed implementations use continuation code 0x4 and Designer code 0x3B.

Note: Non-ARM implementations that follow this CoreSight ID register layout must set the Designer fields appropriate to the implementer.

10.5.2.23.1 SMMU_PMCG_PMAUTHSTATUS

SMMU_PMCG_PMAUTHSTATUS is equivalent to the AUTHSTATUS register that is described in the CoreSight Architecture Specification [8]. It is IMPLEMENTATION DEFINED whether a PMCG supports an authentication interface, as described in [8].

A PMCG might also provide IMPLEMENTATION DEFINED facilities to limit visibility of PMCG events relating to Secure and Non-secure streams, independent of the CoreSight authentication interface.

10.6 Support for two Security states

Security support in a PMCG is optional. If supported, it comprises:

- Whether the PMCG registers can be accessed by both Non-secure and Secure accesses, or only Secure accesses.
- Whether the counters can observe events associated with Secure StreamIDs or Secure state.
- Whether an MSI is sent with an NS == 0 or NS == 1 attribute.

The SMMU_PMCG_SCR.NSRA flag controls whether Non-secure accesses can be made to the registers of a given group. Secure accesses can always be made to the registers.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: As counter groups might exist in separate components, the SMMU does not contain a centralised mechanism for assigning counter groups to a Security state.

The SMMU_PMCG_SCR.SO flag controls whether counters can observe events associated with Secure state or Secure StreamIDs.

When SMMU_PMCG_SCR.SO == 1, Secure observation is enabled and:

- Counters using event types that can be filtered by StreamID might count events that are associated with a Secure or Non-secure StreamID. See section 10.4 and the SMMU_PMCG_EVTYPERn.FILTER_SEC_SID bit.
- Counters using other event types are able to count events that are associated with Secure or Non-secure state, as appropriate for the event.

  Note: IMPLEMENTATION DEFINED events might be associated with StreamIDs or a security state even if they might not be directly associated with a specific StreamID. The architected global events are not specific to a security state.

When SMMU_PMCG_SCR.SO == 0, Secure observation is disabled and:

- Counters using event types that can be filtered by StreamID only count events associated with Non-secure StreamIDs.
- Counters using other event types only count events that are associated with the Non-secure state.

When a counter uses an event type able to be filtered by StreamID but the StreamID filter matches all, events from all StreamIDs associated with the Security state selected by the SMMU_PMCG_EVTYPERn.FILTER_SECSID bit are counted, if permitted by the 'SO' flag.

If a counter group supports MSIs:

- If Security is not supported in the PMCG, the NS attribute of the MSI is 1.
- If Security is supported in the PMCG, the NS attribute of the MSI is given by:

  `NS = SMMU_PMCG_SCR.NSMSI | SMMU_PMCG_SCR.NSRA;`

See section 10.5.2.12.

If a PMCG does not support two Security states but the system does, then the PMCG only observes events that are associated with Non-secure streams. If a PMCG supports two Security states but the system supports one, then all streams are considered to be Non-secure.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

11 Debug/Trace

The SMMU architecture does not require the provision of debug support features. An implementation might provide its own debug features, but if such an implementation supports two Security states a Non-secure debug agent must not be able to access any facilities related to secure transaction handling.

ARM recommends that an implementation provides an IMPLEMENTATION DEFINED mechanism to read the contents of translation and configuration cache structures. If such a mechanism is provided and an implementation supports two Security states, a Non-secure agent must not be able to access Secure cache entries.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

12 Reliability, Availability and Serviceability (RAS)

Note: In this chapter, the term RAS fault is used to distinguish a fault in the sense defined by the RAS architecture from an SMMU translation fault.

The SMMU might support optional RAS features as part of the overall ARM RAS Architecture, which is described in Appendix - The ARM RAS architecture. When supported, RAS fault handling registers according to the ARM RAS Architecture are present, into which errors are recorded.

If the SMMU does not support RAS but the system does, it is possible for the SMMU to consume an external error which is presented to the SMMU driver software as an external abort.

The SMMU might, but is not required to, record errors that it does not itself consume or detect, such as those reported directly from the system to a client device, for example, a device reading a memory location that translates without issue, but ultimately causes the device to consume poisoned data.

12.1 Error propagation, consumption and containment in the SMMU

This guidance section relates SMMU-specific concepts to the RAS architecture.

Generally, SMMU activity can be considered to be driven on demand from incoming transactions from client devices. Consequently, an external transaction might cause the SMMU to consume errors from:

1. Internal state errors.
2. External state errors, for example reading a translation table entry, configuration structure or command with an associated error.

If the SMMU consumes an error while performing translation for an external transaction, containment of the error can be achieved by deferring the error to the source of the transaction. This might involve terminating the transaction with an abort, or otherwise returning error information to the client. A detected error that is silently propagated might lead to silent data corruption (SDC). The termination of a transaction that could be affected by an SMMU error ensures isolation of the error and represents a non-silent error propagation from the SMMU to the client device.

When the SMMU consumes an error from either internal or external state, ARM expects the implementation to report the error and enter error recovery in addition to containment by deferring the error.

Note: An example of an error propagation on write is a write of queue records affected by the error. The error might be propagated with affected data. An implementation might non-silently propagate the error so that the system can poison the data in memory.

A UE can be consumed by the SMMU in a way that can affect external or internal state. This includes:

1. Reading a register containing a UE in order to calculate an address for access to a queue entry.
2. Reading a register containing a UE in order to write the data of a written queue entry.
3. Consuming a UE from the system when reading from the Command queue.
4. Reading a cache entry containing a UE in response to an incoming transaction.
5. Reading a register containing a UE in response to an incoming transaction.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

6. Consuming a UE from the system when fetching data into a cache in response to an incoming transaction.

Containment of these errors avoids silently affecting external or internal state. The effect of the error means, for the corresponding scenarios in the previous list:

1. Internal consistency is lost.
2. The error might be transient (for example, it is overwritten when the queue entry is written), in which case internal consistency is not lost, The error is deferred into the memory system (error on write) and an implementation might record the error. If the error is non-transient, internal consistency might be lost.
3. An implementation might have unsuccessfully tried to correct the error, or might accept the error as a UE. However, internal consistency is not lost.
4. The transaction (and therefore other agents in the system) would be affected, and isolation broken, unless the transaction is terminated. The UE can therefore be deferred to the client. SMMU internal consistency might not be lost.
5. An error in a register might not be correctable. The UE can be deferred to the client and recorded. Internal consistency might have been lost if the register could affect future transactions.
6. The transaction would be affected by the error, unless the transaction is terminated (deferring the UE to the client). SMMU internal consistency might not be lost.

If internal consistency is lost, invoking a Service Failure Mode (SFM) can isolate further errors from the system. This might reduce the severity of the error by ceasing subsequent duplicate error reports caused by the same failure, and by reducing the chance that the loss of internal consistency can silently propagate an error.

12.2 Error consumption visible through the SMMU programming interface

On consumption of an external error, an SMMU supporting RAS is expected to report a RAS event through the RAS register interface. Alternatively, consumption of an external error by an SMMU that does not support RAS presents it with an external abort. In both cases, the SMMU records the event through the Event queue (pertaining to the Security state of the transaction that caused the error to be consumed) or GERROR as appropriate:

- F_WALK_EABT where a translation table walk consumed an error.
- F_STE_FETCH where an STE fetch consumed an error.
- F_CD_FETCH where a CD fetch consumed an error.
- SMMU_(S_)GERROR errors:
    - GERROR.CMDQ_ERR triggered and CERROR_ABT reported in SMMU_(S_)CMDQ_CONS.ERR, where a command fetch consumed an error.
    - GERROR.PRIQ_ABT_ERR triggered when a PRI queue access is aborted because of an external error.
    - GERROR.EVENTQ_ABT_ERR triggered when an Event queue access is aborted because of an external error.

12.3 Service Failure Mode (SFM)

If internal consistency is known to have been lost (for example, detection of a UE in internal register state), or there is a likelihood that it has been lost, and the functionality of the SMMU will be impaired or would risk silent data corruption or silent propagation of errors, Service Failure Mode must be entered. The SMMU must terminate

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

client transactions after this mode is entered, and stop accessing its queues. ARM recommends that the SMMU registers are still be readable in this mode to aid diagnosis. The mechanism to exit or recover from this mode is IMPLEMENTATION DEFINED, but must include system reset.

Note: An implementation might provide a partial service failure mode, which depends on implementation specific isolation features or safety guarantees, for example, a partitioning system in which some client devices are guaranteed to be unaffected by a loss of consistency in a different portion of the SMMU and whose traffic might still continue to flow.

Entry to Service Failure Mode is signalled by all of the following means:

- The Global Errors SMMU_GERROR.SFM and SMMU_S_GERROR.SFM are triggered for both Non-secure and Secure programming interfaces (if implemented).
- An IMPLEMENTATION DEFINED notification such as recording syndrome into RAS registers and asserting an Error Recovery Interrupt or system-wide error interrupt.

Diagnosis of the reason for entering SFM is made through IMPLEMENTATION DEFINED means.

12.4 RAS fault handling/reporting

When RAS facilities are implemented, an implementation must provide at least one group of memory-mapped error recording registers in accordance with the RAS fault handling register format defined in Appendix - The ARM RAS registers.

The exact layout and operation of the RAS registers is IMPLEMENTATION DEFINED, including:

- Discovery and identification registers.
- The number of RAS register groups and association to nodes.
- Whether counters are implemented or single errors reported.

Error Recovery Interrupts and Fault Handling Interrupts must be provided.

The mechanism for determining whether RAS facilities are implemented, base addresses for RAS registers and the extent of RAS register frames is IMPLEMENTATION DEFINED. Note: For example, IMPLEMENTATION DEFINED identification registers or firmware descriptions.

When Security support is implemented, one RAS register interface might be provided for each of the Secure and Non-secure programming interfaces, only Secure or only Non-secure. When Security support is not implemented, a RAS register interface accessible by Non-secure accesses is provided.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

13 Attribute Transformation

The ARM architecture supports a set of attributes that accompanies each external access that is made by a PE. These attributes map onto attributes carried by the interconnect in an interconnect-specific manner. The SMMU supports a set of attributes that is identical to that supported by the PE. These attributes govern the behavior of each client transaction and are presented from the SMMU to the memory system. The interconnect from the client device might also present attributes to the SMMU. This section describes how the attributes supplied with an input transaction affect the flow through the SMMU and how the attributes output with a transaction are determined.

To avoid interconnect-specific language, this section uses attribute terminology and concepts as defined in the ARM Architecture Reference Manual [4]. The mapping of these attributes onto interconnect implementations that support a subset of the attributes is IMPLEMENTATION DEFINED.

This section covers the attributes that are applied in the four conditions under which a transaction can pass through an SMMU:

1. Global bypass. The SMMU is disabled (SMMU_(S_)CR0.SMMUEN == 0).
2. STE bypass. The SMMU is enabled but all stages of translation are disabled by an STE.
3. Normal translation flow. An STE configures one or more stages of translation.
4. PCIe ATS Translated. A device requests a ‘pre-translation’, caches the result, and subsequently presents a transaction with an address that has been translated but has not had SMMU attributes applied.

This section additionally covers accesses that originate from the SMMU, such as fetching configuration or walking translation tables.

Depending on the path taken, attributes might be overridden or modified by SMMU configuration registers, STE fields or translation table entries. Some of the attributes are used when checking access permissions in translation table entries, others are provided for the memory system.

At the highest level, the attribute determination for the normal translation flow is:

1. Input attributes are obtained from the incoming transaction and any attributes that are not provided are generated from constant default values.
2. The STE might then override incoming attributes for a specific stream.
3. The stage 1 and stage 2 translation table permissions are checked against the attributes determined from the previous step, where a stage is enabled.
4. If stage 1 is enabled, the stage 1 translation table attributes replace any input and STE override values.
5. If stage 2 is enabled, the stage 2 translation table attributes are combined with the stage 1 attributes if stage 1 is enabled, or combined with the overridden input attributes if stage 1 is not enabled.
6. The attribute is output with the translated transaction.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

13.1 SMMU handling of attributes

13.1.1 Attribute definitions

A transaction has the attributes that are defined by the ARM architecture [4]:

Attribute	Shorthand	Values
Memory Type	MT	Normal i{WB, WT, NC}-o{WB, WT, NC} – generally Normal, Device-{GRE, nGRE, nGnRE, nGnRnE} – generally any-Device.
Shareability	SH	Non-shareable (NSH),Inner shareable (ISH), Outer shareable (OSH) Note: Device types and Normal iNC-oNC are considered to be OSH
Read Allocation hint	RA	Inner read allocate/no-allocate, Outer read allocate/no-allocate
Write Allocation hint	WA	Inner write allocate/no-allocate, Outer write allocate/no-allocate
Transient hint	TR	Inner transient/non-transient, Outer transient/non-transient
Read/Write	R/W	Read, Write
Inst/Data	INST	Instruction, Data
Privilege	PRIV	Privileged/Unprivileged
Non-secure	NS	Secure, Non-secure

Key: i = Inner,
o = Outer,
WB = Write-Back cacheable,
WT = Write-Through cacheable,
NC = Non-Cacheable,

In this document, a Normal memory type is expressed as:

- iNC-oNC, or
- i{ NC, {WB, WT}/[n]RA[n]WA[n]TR }-o{ NC, {WB, WT}/[n]RA[n]WA[n]TR }-{NSH,ISH,OSH}
  (Syntax: {a,b} = Choose one of a or b, [x] = x is optional.)

This means that:

- Normal-iWB/RAnWATR-oNC-ISH:
  - Is Inner writeback cacheable/Read-Allocate/no-Write-Allocate/transient, outer non-cacheable inner-shareable.
  - The (outer) NC type has no RA/WA or TR attributes.
- Normal-iWB/RAWAnTR-oWB/RAWAnTR-OSH:
  - Is inner/outer writeback read/write allocate non-transient, outer-shareable.
- Normal-iNC-oNC
  - Is architecturally non-cacheable, always OSH
  - NC has no RA/WA or TR attributes.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: System shareable is an AMBA interconnect term and is not used in this document. Device memory is OSH according to the ARM architecture, but the AMBA interconnect Device type is System shareable. Similarly, a PE Normal-iNC-oNC attribute becomes Non-Cacheable System shareable in AMBA interconnect. ARMv8-A memory attribute terminology is used in this document.

While some interconnects might support an IMPLEMENTATION DEFINED subset of these attributes, R/W is mandatory.

The R/W, INST and PRIV attributes are used to check translation permissions against TTD read/write and execute access permissions. Some memory systems might also use INST, PRIV, RA, WA and TR attributes to influence cache allocation or prioritisation policies but such usage is outside the scope of this document.

When the SMMU supports two Security states, NS is used by the memory system to distinguish between Secure physical addresses and Non-secure physical addresses, and to check against SMMU_S_CR0.SIF.

Some interconnects support atomic transactions. These are treated as performing both a read and a write for permission checking purposes. For Event records generated due to atomic accesses:

- In the event of a permission fault due to having write access but not read access, the value of the F_PERMISSION event's RnW field is:
  - SMMUv3.0: IMPLEMENTATION DEFINED.
  - SMMUv3.1: 1.
- Note: In the event of a permission fault in response to not having write access, or having no access, the F_PERMISSION event is recorded with RnW == 0.
- For all other causes that trigger F_PERMISSION, and for all other event types, RnW == 0.

The access permission attributes of an incoming transaction are only used by the SMMU to enforce access permissions against the permissions determined by translation table descriptors and SMMU_S_CR0.SIF. They do not interact with the memory type in any way.

Note: In AArch32 state, a PE is permitted to raise a Permission fault when an instruction fetch is made to Device memory and execute-never is not set. The SMMU does not raise a Permission fault in this case, and the output attribute remains Device. An access is permitted if the permission checks succeed, regardless of the final memory type.

13.1.2 Attribute support

The SMMU defines behavior for all architected attributes. However, where an interconnect from the SMMU to the memory system supports only a subset of these attributes, the SMMU is not required to generate attributes that it does not use. With the exception of R/W, INST, and PRIV all configuration fields that affect unused attributes are IGNORED.

When SMMU_S_IDR1.SECURE_IMPL == 1:

- Override fields that are associated with NS might be supported.
- Any transaction that belongs to a Non-secure stream is outputted with NS == 1.

When SMMU_S_IDR1.SECURE_IMPL == 0:

- Override fields that are associated with NS are not supported.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Where supported by the memory system, all transactions are outputted with NS == 1.

If an interconnect supports attributes but does not differentiate inner and outer in the same way as the architecture, the mapping onto the architecturally-defined set of attributes is IMPLEMENTATION DEFINED.

In some implementations, a client device might request translation services from an SMMU and later perform transactions separately, based on the translations and attributes determined by the SMMU. The behavior of the client device transactions with respect to attributes presented to the system are outside the scope of the SMMU architecture. Where such a client device does not support certain memory types or attributes that are returned by the SMMU translation process, behavior on accesses with these memory types is defined by the device implementation. However, the page permission protection model of the SMMU translations must always be fully observed.

Note: For example, a client device must never use a read-only translation to issue a write transaction into the system. If a client device does not support a particular memory type, a valid behavior might be to alter the type in an architecturally-safe manner. This could be strengthening a Device-GRE access to a Device-nGnRnE access. If this is not possible, another valid behavior might be for the client device to abort the access in a client device-specific manner.

Note: A composite device that contains an SMMU, such as an intelligent accelerator, is conceptually identical to a client device making requests of an external SMMU.

The SMMU makes its own accesses to memory, for example to fetch configuration or perform translation table walks. Each access has a memory type and attributes that are configured using SMMU registers or structure fields. Where an SMMU and the memory system support only a subset of memory types, and where access to an unsupported type can be reported to the SMMU, the SMMU will record an external abort for the cause of the access in question in the architected manner. If such an access is instigated by an incoming transaction, the transaction will be terminated with an abort.

A fetch from unsupported memory types generates the following fault and errors:

- STE: F_STE_FETCH
- CD: F_CD_FETCH
- Command queue read entry: GERROR.CMDQ_ERR and Command queue CERROR_ABT
- Event queue access: GERROR.EVENTQ_ABT_ERR
- PRI queue access: GERROR.PRIQ_ABT_ERR
- MSIs: GERROR.MSI_*_ABT_ERR
- Translation table walk: F_WALK_EABT

ARM strongly recommends that an SMMU supports all architected access types (for both its own structure accesses and for client transactions) so that it is compatible with generic driver software.

The SMMU considers all incoming writes to be Data, even if the client device marks the incoming write as Instruction. This is done on input, prior to any translation table permission checks. In addition, STE.INSTCFG and SMMU_(S_)GBPA.INSTCFG can only change the instruction/data marking of reads. Where an implementation outputs an INST attribute to the memory system, this reflects the output of the INSTCFG fields for reads and is fixed as DATA for writes.

All reads that originate from the SMMU, for example structure fetch, are represented as Privileged-Data if INST/PRIV attributes are supported by the memory system.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

All SMMU writes of output queue data and MSIs are represented as Privileged-Data if INST/PRIV attributes are supported by the memory system.

13.1.3 Default input attributes

Where the SMMU is not supplied with an attribute because the interconnect between the client device and SMMU does not have the ability to convey it, the SMMU constructs a default input value:

Attribute	Input generated as:
MT	Normal iWB-oWB
SH	NSH
RA	Allocate
WA	Allocate
TR	Non-transient
INST	Data
PRIV	Unprivileged
NS	Non-secure

These values are used when any configuration path is set to 'Use incoming' attribute, but the attribute is not supplied.

Note: The SMMU architecture only includes configuration of inner and outer properties for the Cacheability of a memory type. No separate inner or outer configuration is provided for RA, WA or TR in any location aside from the translation table descriptor. Attribute overrides for RA, WA and TR affect both inner and outer allocation and transient hints at the same time.

13.1.4 Replace

The Replace operation discards any input attribute, replacing a specific attribute with a configured value. For example, the SMMU_GBPA.SHCFG field allows the input Shareability to be discarded and replaced by NSH/ISH/OSH values, or used directly without override, for the purposes of global bypass when the SMMU is disabled.

In this document, the term Override refers to a configuration field that causes an attribute to be replaced with a specific value.

The STE fields INSTCFG, PRIVCFG, NSCFG, SHCFG, ALLOCCFG, MTCFG/MemAttr contain override fields that can cause individual input attributes to be replaced with values given in these fields. Similar fields in SMMU_(S_)GBPA allow override of attributes when transactions globally bypass. Whether overrides in these fields take effect is indicated by SMMU_IDR1.ATTR_TYPES_OVR and SMMU_IDR1.ATTR_PERMS_OVR. See 6.3.2 and the field descriptions in 5.2 for more information.

When stage 1 translation is used, memory type attributes that are provided by translation table entries replace the attributes provided by the input transaction and any input overrides, if appropriate.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

This operation is represented by `replace_attrs()` in pseudocode.

Global or STE input overrides are not be permitted to make output attributes inconsistent. Because allocation and transient hints are only relevant to Normal cacheable memory types, their overrides in either SMMU_(S_)GBPA or in the STEs do not affect a transaction with the Device or Normal iNC-oNC type. The override value is ignored. Similarly, an input type of Device or Normal-iNC-oNC, whether supplied by the client device or overridden in SMMU_(S_)GBPA or in the STE, will always be OSH regardless of any SHCFG override.

When an input transaction provides a memory type that is not cacheable and an STE or GBPA input override changes the inner or outer type to a cacheable type, the input RA, WA and TR attributes are taken to be 'RA, WA, nTR' (consistent with 13.1.3), unless these attributes are also overridden.

13.1.5 Combine

The result of combining one attribute value with another is governed by rules that are similar to those that apply to SMMUv2 and the ARM Architecture's combining of stage 1 and stage 2 attributes. The SMMUv3 behavior is the same as that of a PE in that stage 2 translation, when enabled, combines its attribute with those from stage 1, or with the incoming attributes when stage 1 is not enabled.

This operation is represented by `combine_attrs()` in pseudocode.

Note: The SMMUv3 behavior with respect to Read-Allocate, Write-Allocate and transient hints with stage 2 differs from SMMUv2 in that SMMUv3 has no stage 2 overrides of RA/WA/TR. However, an order of precedence is defined here to make the pseudocode clearer.

The permission-related attributes (R/W, INST, PRIV, NS) are not subject to combine operations. These attributes are only used in the SMMU to check against each enabled stage of translation descriptor permissions. Only MT, SH, RA/WA and TR are combined.

For any one attribute, an order of strength is observed. For each attribute, a combine operation takes the stronger of the values presented.

Weakest						Strongest
Normal-WB	**Normal-WT**	**Normal-NC**	**Device-GRE**	**Device-nGRE**	**Device-nGnRE**	**Device-nGnRnE**

Weakest		Strongest
NSH	**ISH**	**OSH**

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Weakest	Strongest
Allocate	**No-allocate**

Weakest	Strongest
Non-transient	**Transient**

The order for allocate and transient hints is chosen so that allocate and non-transient are the normal cases that are pulled down for no-allocate or transient special cases.

13.1.5.1 Combine examples

Normal-iWB/RAWAnTR-oNC-ISH + Device-nGnRE = Device-nGnRE

Device-nGnRE + Device-nGnRnE = Device-nGnRnE

Normal-iWB/RAWAnTR-oNC-ISH + Normal-iWT/RAWAnTR-oWT/RAnWATR-OSH

= Normal-iWT/RAWAnT-oNC-OSH

(Note: outer non-transient even though onTR+oTR = oTR, as outer non-cacheable – see below.)

13.1.6 Ensuring consistent output attributes

The SMMU does not allow the output of inconsistent attribute combinations. If overrides, translation table configuration or bad input results in inconsistent attributes, the SMMU ensures consistency:

- Any-Device and Normal iNC-oNC are output as Outer Shareable
- For Normal types, NC at either inner or outer cache level have no RA/WA or TR attributes at that level. A non-cached access is considered to be read-no-allocate, write-no-allocate, non-transient, regardless of programmed read/write allocate and transient configuration.
- For Normal types, a cacheable type that is read-no-allocate and write-no-allocate has no TR attribute. Such a type is considered to be non-transient, regardless of any input or programmed override configuration.

This also applies to the result of ATOS translation operations. See section 9.

Note: In addition to these architectural attribute consistency rules, an implementation might include interconnect-specific consistency rules. See section 16.7.5.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

13.2 SMMU disabled global bypass attributes

When the SMMU is disabled for a given Security state, transactions that belong to that Security state are subject to global bypass attribute overrides.

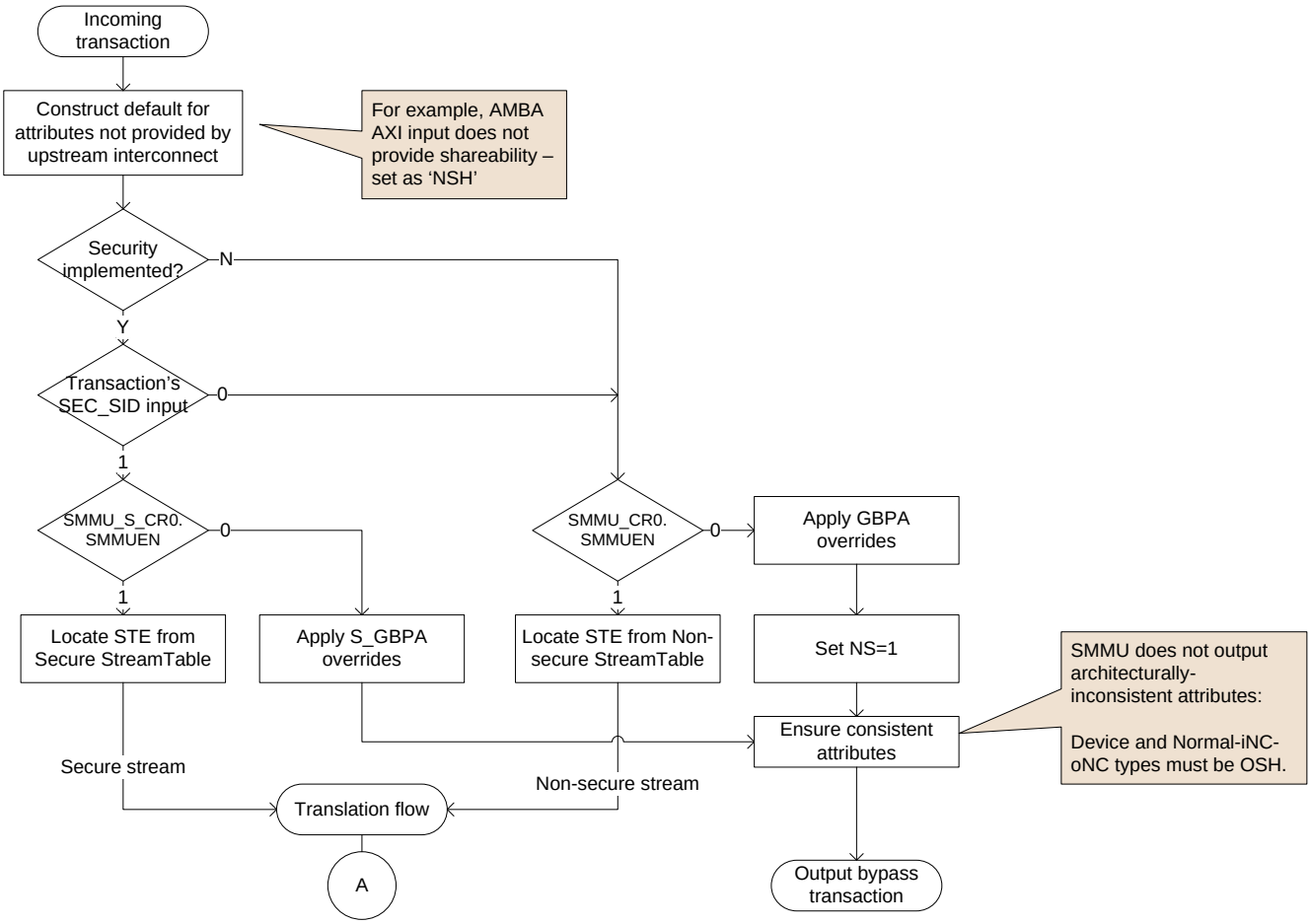


Figure 20: Input attribute flow - SMMU-disabled bypass and start of translation flow

When SMMU_S_IDR1.SECURE_IMPL == 0 and SMMU_CR0.SMMUEN == 0, all transactions bypass the translation and configuration facilities of the SMMU. The memory type, Shareability, Read/Write allocate, Transient, Inst/Data and Privilege attributes of the transaction might be individually replaced by the MTCFG/MemAttr, SHCFG, ALLOCCFG, INSTCFG and PRIVCFG override fields of the SMMU_GBPA register, respectively. Each attribute might take a set value or select to Use incoming. In the latter case, the attribute is either taken from the incoming interconnect or, when not supplied, the default input attribute constructed in the SMMU. Apart from these overrides, the transaction is output directly.

Alternatively, when SMMU_S_IDR1.SECURE_IMPL == 0 and SMMU_CR0.SMMUEN == 1, an STE is located and the translation flow begins.

When SMMU_S_IDR1.SECURE_IMPL == 1, the behavior of a transaction on arrival at the SMMU is determined by the Security state of its StreamID, as determined by the SEC_SID input, see section 3.10.1. If the stream is Non-secure and SMMU_CR0.SMMUEN == 0, the attributes are overridden by SMMU_GBPA as described earlier in this section. If the stream is Secure and SMMU_S_CR0.SMMUEN == 0, the attributes are determined by SMMU_S_GBPA in the same way, otherwise a secure STE is located and the translation flow begins. The output NS attribute is determined by SMMU_S_GBPA.NSCFG, it can use the incoming value or be overridden to either state.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The reset state of the bypass attribute register SMMU_GBPA, and when SMMU_S_IDR1.SECURE_IMPL == 1 SMMU_S_GBPA, is IMPLEMENTATION DEFINED.

Note: The SMMU might provide an implementation-time configuration of the default bypass attributes which allows a system designer to ensure that device DMA has useful attributes even if system software has not initialized the SMMU.

ARM recommends that the default attributes are set as appropriate for the system in accordance with relevant base system architecture definitions.

This case is illustrated by this pseudocode:

```
// See 13.1.3:
Attrs input_attrs = add_defaults_for_unsupplied(upstream_attrs);

bool secure = (SEC_SID == 1);

if (secure) {
      output_attrs = replace_attrs(SMMU_S_GBPA, input_attrs);
} else {
      output_attrs = replace_attrs(SMMU_GBPA, input_attrs);
      output_attrs.NS = 1;
}
// MT, SH, RA, WA, TR, INST, PRIV are unchanged or overridden,
// depending on respective register field.
//
// NS is unchanged or overridden if secure stream, otherwise
// fixed at 1.

// See 13.1.6:
output_attrs = ensure_consistent_attrs(output_attrs);
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

13.3 Translation flow, STE bypasses stage 1 and stage 2

A second type of SMMU bypass is available when the SMMU is enabled. This is when a StreamID selects an STE configured to bypass (STE.Config == `0b100`). SMMU_GBPA and SMMU_S_GBPA are not used, and a finer-grained per-STE configuration is available. Refer to the no-translate case of Figure 21.

The STE fields MTCFG, MemAttr, ALLOCCFG, SHCFG, NSCFG, PRIVCFG and INSTCFG can override any attribute before the transaction is passed to the memory system. This provides per-StreamID granularity of attribute settings.

Each attribute might take a set value or select to Use incoming. Where the incoming attribute is use, the attribute is either taken from the incoming interconnect or, when not supplied, the default input attribute constructed in the SMMU.

Pseudocode:

```
// See 13.1.3:
Attrs input_attrs = add_defaults_for_unsupplied(upstream_attrs);

STE ste = get_STE_for_stream(StreamID);

output_attrs = replace_attrs(ste, input_attrs);
if (!ste.fetched_for_secure_stream()) {
      output_attrs.NS = 1;
}
// MT, SH, RA, WA, TR, INST, PRIV are unchanged or overridden,
// depending on respective STE field.
//
// NS is unchanged or overridden if secure stream, otherwise
// fixed at 1.

output_attrs = ensure_consistent_attrs(output_attrs);
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

13.4 Normal translation flow

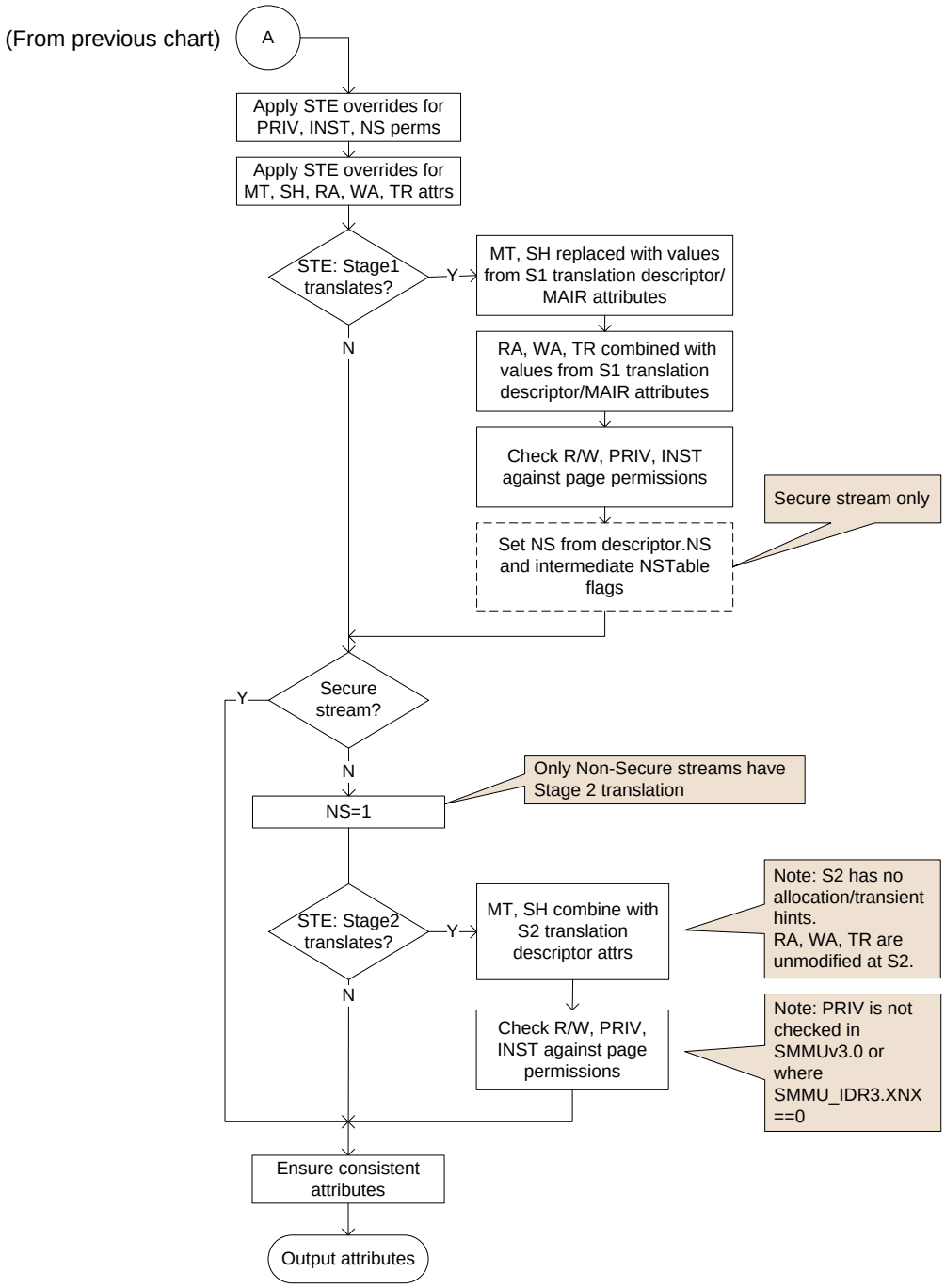


Figure 21: Attribute determination in translation

When the StreamID of a transaction selects an STE configuration with two stages of translation, the memory attribute of the transaction is determined by the stage 1 TTD attributes, which are determined using the MAIR fields, and combined with the stage 2 TTD attributes. For stage 1-only translations, the memory attributes are determined solely by the stage 1 TTD attributes. For stage 2-only translations, the memory attributes are determined by the input attribute, with STE overrides, combined with the stage 2 TTD attributes.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

First, the incoming attributes, with defaults constructed for any unsupplied attributes, are subject to the STE override described in section 13.3. These attributes are provided to the first enabled stage of translation. If stage 1 bypasses, these attributes are provided to stage 2 directly.

If stage 1 translates, the stage 1 TTD, in combination with the MAIR fields of the CD, determine the stage 1 attributes and page permissions. The R/W, PRIV and INST attributes,(as modified by STE overrides, are checked against the page permissions.

13.4.1 Stage 1 page permissions

The effect of the lower bit of the stage 1 translation table descriptor AP[2:1] field, AP[1], changes depending on the translation regime under which the translation is performed, as determined by StreamWorld. For Secure or Non-secure EL1 or for EL2-E2H, the AP[1] bit controls the EL0 access permissions of the page (PRIV == 0). However, when StreamWorld == EL2 or StreamWorld == EL3, the AP[1] bit is ignored and treated as if it were 1. This has the same effect as treating PRIV == 1 for the purposes of permission checking, which means that input PRIV attributes and any STE.PRIVCFG overrides are ignored for these checks.

When PRIV and INST attributes are output to the memory system, they are presented as the result of applying the STE.PRIVCFG and STE.INSTCFG overrides to the input attribute. They are not modified after this point.

13.4.2 Stage 1 memory attributes

The attributes determined from the stage 1 translation table descriptor and CD.MAIR are provided directly to the stage 2 translation. The incoming attributes, including any STE overrides, are discarded and replaced by the stage 1 attributes, with the exception of RA/WA/TR.

When the translation is from a Secure StreamID, the NS attribute that is output is determined directly from the effective NS bit of the descriptor, which must take into account the value of NSTable flags on intermediate table descriptors in addition to the final descriptor NS flag, in the same way as on the PE. In addition, CDs that are used with Secure streams contain NSCFGx flags that determine the starting NS attribute of translation table walks performed through CD.TTB0/1. When a translation is fetched using a CD.TTB0/1 with a corresponding NSCFGx == 1, the effective final NS bit is always 1. When a translation is from a Non-secure stream, the output NS attribute that is output is always 1 and the NS attribute that was input with the transactions is ignored.

Note: The behavior of NS matches the SMMUv2 behavior in that it is wholly determined by the stage 1 descriptor and translation table walk configuration. The input NS attribute, and STE override, only affect the output NS attribute when no stage 1 translation is configured.

When the attribute input to stage 1 is a Normal-cacheable type, the RA/WA/TR attributes are not discarded and instead are combined with those determined from the stage 1 translation TTD and MAIR. Any other type does not communicate RA/WA/TR hints, in which case the stage 1 translation RA/WA/TR attributes are used directly. All other attributes must be overridden by stage 1 translation table attributes.

Note: This property, combined with STE.ALLOCCFG == `0b0xxx` configuration, allows a client device to influence per-transaction allocation/transient attributes when using stage 1 translation tables, allowing sub-page control.

13.4.3 Stage 2

If stage 2 bypasses, the attributes provided by the incoming transaction or the stage 1 translation are forwarded unchanged to the memory system.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If stage 2 translates, the MT and SH attributes input to stage 2 undergo a combine operation with the stage 2 translation descriptor attributes and the result passed on.

Stage 2 descriptors do not include read/write allocate or transient hints so the RA/WA/TR attributes provided to stage 2 are passed on unchanged.

The INST attribute is checked, along with R/W, against the descriptor page permissions. In SMMUv3.0 implementations or where SMMU_IDR3.XNX == 0, the PRIV attribute is not used to evaluate stage 2 permissions. Where SMMU_IDR3.XNX == 1, the PRIV attribute is used to evaluate stage 2 permissions. The values of INST and PRIV are passed on unchanged.

13.4.4 Output

The transaction is output after ensuring consistency of the attributes and forcing NS == 1 for transactions on Non-secure streams.

Pseudocode, which also illustrates the cases describe in sections 13.2 and 13.3 in more detail, is show below:

```
// See 13.1.3:
Attrs input_attrs = add_defaults_for_unsupplied(upstream_attrs);

bool secure = (SEC_SID == 1);

if (secure && SMMU_S_CR0.SMMUEN == 0) {
      output_attrs = replace_attrs(SMMU_S_GBPA, input_attrs);
      // See 13.1.6:
      output_attrs = ensure_consistent_attrs(output_attrs);
      output_transaction(output_attrs);   // Address, direction not shown
      // Done.
} // Else carry on below

if (!secure && SMMU_CR0.SMMUEN == 0) {
      output_attrs = replace_attrs(SMMU_GBPA, input_attrs);
      output_attrs.NS = 1;
      // See 13.1.6:
      output_attrs = ensure_consistent_attrs(output_attrs);
      output_transaction(output_attrs);   // Address, direction not shown
      // Done.
} // Else carry on below

// If we get to here, the appropriate SMMU interface is enabled and
// configuration can be fetched:
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

```
STE ste = get_STE_for_stream(StreamID);

Attrs i;

// Starting attributes are as input, with STE overrides:
i = replace_attrs(ste, input_attrs);

if (ste.s1_translates()) {
      Attrs s1_attrs;
      TTD   s1_ttd;

      CD cd = get_CD(ste, StreamID, SubstreamID, SubstreamID_valid);

      s1_ttd = s1_translation(cd, ste);
      // Address not shown; IPA determined from VA.

      s1_attrs = s1_ttd.lookup_attrs();
      // The returned s1_attrs contains attributes determined from the
      // translation descriptor and MAIR, including RA/WA and TR.

      if (s1_ttd.translation_fault()) {
            // XXXX Stage 1 translation fault

            // Note: A speculative read transaction is simply terminated
            // with abort here, without recording an event.
      }

      // Note: Permission checking must consider HTTU configuration,
      // and update Access/Dirty flags instead of signaling a related fault
      // when HTTU enabled:
      if (check_perms_and_httu(cd, s1_ttd, i.RW, i.INST, i.PRIV) == FAULT) {
            // XXXX Stage 1 permissions or Access flag fault

            // Note: EL2 or AArch64 EL3 stage 1 do not check PRIV,
            // i.e. client traffic behaves as if PRIV == 1.  However,
            // the PRIV attribute is not modified.
      }

      // Note: See section 13.4.2; when a cached type is input to Stage 1,
      // s1_attrs.{RA,WA,TR} are combined with i.{RA,WA,TR} instead of
      // replacement (non-cached/Device types do not express RA/WA/TR
      // attributes):
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

```
        if (is_cached_type(i.MT)) {
              i.{RA,WA,TR} = combine_attrs(i.{RA,WA,TR}, s1_attrs.{RA,WA,TR});
        } else {
              // An input NC/Device type does not provide these attributes,
              // so take these attributes from S1:
              i.{RA,WA,TR} = s1_attrs.{RA,WA,TR};
        }

        // MT and SH are taken directly from Stage 1:
        i.{MT,SH} = s1_attrs.{MT,SH};

        // NS is taken from the ‘effective’ value calculated from
        // S1 page descriptor, CD.NSCONFIGx and appropriate NSTable bits:
        i.NS = s1_ttd.get_effective_NS();

        // RW, INST, PRIV do not change.
  } else {
        // MT, SH, RA, WA, TR unchanged; and RW, INST, PRIV, NS
        // permissions passed through.
  }

  if (ste.s2_translates()) {
        Attrs s2_attrs;
        TTD   s2_ttd;

        s2_ttd = s2_translation(ste);
        // Address not shown; PA determined from IPA and HTTU is performed.

        s2_attrs = s2_ttd.lookup_attrs();

        if (s2_ttd.translation_fault()) {
              // XXX Stage 2 translation fault

              // Note: On any fault, a speculative read transaction is simply
              // terminated with abort.

              // Note: HTTU at Stage 1 is permitted, even if we fault @S2
        }

        // S2 does not encode RA/WA/TR; these values plus combine_attrs() below
        // result in “take input from S1/previous step” behavior for RA/WA/TR:
        s2_attrs.RA_inner = true;
        s2_attrs.RA_outer = true;
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

```
        s2_attrs.WA_inner = true;
        s2_attrs.WA_outer = true;
        s2_attrs.TR_inner = false;
        s2_attrs.TR_outer = false;

        // MT, SH, RA, WA, TR updated via combine:
        i = combine_attrs(i, s2_attrs);

        // See HTTU note in Stage 1:
        if (check_perms_and_httu(ste, s2_ttd, i.RW, i.INST, i.PRIV) == FAULT) {
              // XXXX Stage 2 permissions or Access flag fault

              // Note: S2 checks PRIV if SMMUv3.1 is implemented
              // and SMMU_IDR3.XNX == 1.
        }
  } else {
        // MT, SH, RA, WA, TR unchanged; and RW, INST, PRIV, NS
        // permissions passed through.
  }

  // Finally, a non-secure stream is not permitted to output NS == 0;
  // the overrides and page attribute are ignored:
  if (!ste.fetched_for_secure_stream()) {
        output_attrs.NS = 1;
  }
  // MT, SH, RA, WA, TR may be modified by translations and overrides.
  // INST, PRIV may be modified by STE overrides.
  //
  // For a secure stream, NS is determined from the Stage 1 translation
  // (if present), otherwise is the STE override or input value.

  // See 13.1.6:
  output_attrs = ensure_consistent_attrs(output_attrs);

  output_transaction(output_attrs);   // Address, direction not shown

  // Done.
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

13.5 Summary of attribute/permission configuration fields

The attributes output from the SMMU are determined as follows.

	If translation occurs			If bypass	
Attribute	Stage 1-only	Stage 1 + stage 2	Stage 2-only	SMMUEN == 1 but STE.Config == `0b100`	SMMUEN == 0
INST	STE.INSTCFG				SMMU_(S_)GBPA.INSTCFG
PRIV	STE.PRIVCFG				SMMU_(S_)GBPA.PRIVCFG
NS (Secure stream)	Effective S1_TTD.NS, NSTable and CD.NSCFGx	N/A		STE.NSCFG	SMMU_S_GBPA.NSCFG
NS (Non-secure stream)	Fixed, NS == 1				
MT	CD.MAIR[S1_TTD.AttrIndx]	Combine(CD.MAIR[S1_TTD.AttrIndx], S2_TTD.MemAttr)	Combine(STE.{MemAttr, MTCFG}, S2_TTD.MemAttr)	STE.{MemAttr, MTCFG}	SMMU_(S_)GBPA.{MemAttr, MTCFG}
RA, WA, TR	CD.MAIR[S1_TTD.AttrIndx], if STE.{MemAttr, MTCFG} provide any-Device or NC input to Stage 1, otherwise: Combine(STE.ALLOCCFG, CD.MAIR[S1_TTD.AttrIndx])	CD.MAIR[S1_TTD.AttrIndx], if STE.{MemAttr, MTCFG} provide any-Device or NC input to Stage 1, otherwise: Combine(STE.ALLOCCFG, CD.MAIR[S1_TTD.AttrIndx])	Combine(STE.ALLOCCFG, S2_TTD.MemAttr)	STE.ALLOCCFG	SMMU_(S_)GBPA.ALLOCCFG
SH	S1_TTD.SH	Combine(S1_TTD.SH, S2_TTD.SH)	Combine(STE.SHCFG, S2_TTD.SH)	STE.SHCFG	SMMU_(S_)GBPA.SHCFG

Notes:

- Permission checking of InD and PnU against translation descriptor fields is not shown here. Also, many systems might not require InD/PnU to be output into the system.
- References to INSTCFG, PRIVCFG, NSCFG, MTCFG/MemAttr and ALLOCCFG refer to the effective output of these override fields, which might always be the incoming attribute depending on

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

SMMU_IDR1.ATTR_PERMS_OVR and SMMU_IDR1.ATTR_TYPES_OVR. This table does not show the effect of one field on another field. For example, the configuration used to determine allocation/transient hints or Shareability is irrelevant if the MemType is of any-Device type. See section 16.7.5 for other interconnect-specific dependencies between attributes.

13.6 PCIe and ATS attribute/permissions handling

This section covers an area of system architecture that intersects with the SMMU operation. Integration of PCIe, particularly with ATS, requires consideration of the presentation of transactions to the SMMU from the Root Complex and the overall system architecture, including platform definitions such as Server Base System Architecture [10], in addition to that of the SMMU itself.

In many systems it will be typical to have a discrete Root Complex IP block connected to a downstream SMMU IP block by a standard piece of interconnect fabric, for example using AXI. This section considers the attributes provided to the SMMU by the Root Complex on such an upstream interconnect in relation to the output attributes provided by the SMMU downstream into the system.

13.6.1 PCIe memory type attributes

PCIe does not contain memory type attributes, and each transaction takes a system-defined memory type when it progresses into the system. The Server Base System Architecture [10] requires the base memory type to be cacheable shared (IO-coherent).

Note: When PCIe is used with an SMMU, the SMMU can assign a different attribute if necessary, but the common usage model is for PCIe DMA to remain cacheable shared and the SMMU is primarily used for address transformation/protection rather than attribute assignment.

Note: SBSA requires that, when no SMMU is present or an SMMU is present and in global or stream bypass mode, a transaction from a Root Complex targeted at memory is presented to the memory system with a fixed attribute of cached, shared. The exact attribute will be Normal-iWB-oWB-ISH or OSH and is system-dependent, but is considered fixed/static because PCIe does not encode different memory types. The system must treat transactions that are targeted at devices (such as MSIs) as Device type accesses.

When an SMMU is used with translation (i.e. not in bypass mode), its configuration might use Stage 1/Stage 2 translation and/or STE input overrides, determining an output attribute from a function of the input, TTD and override attributes as described previously in this chapter. The final attribute is defined by software configuration.

Note: See No_snoop below, which might alter the output final attribute.

Note: See SMMU_GBPA (6.3.13) and STE.{MTCFG,ALLOCCFG,SHCFG}: it is IMPLEMENTATION DEFINED whether attribute overrides affect streams associated with PCIe devices or whether the incoming attribute is used. ARM does not expect attribute overrides to be required for PCIe devices.

13.6.1.1 No_snoop

Note: In PCIe, No_snoop is not guaranteed to be supported by an endpoint or Root Complex, and requires explicit enablement via configuration space. When a transaction includes a No_snoop == 1 flag, it indicates that the transaction is allowed to “opt-out” of hardware cache coherency; software cache coherency ensures the access would not hit in a cache, so the I/O access is permitted to avoid snooping caches.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

In an ARM system, a No_snoop access corresponds to Normal-iNC-oNC-OSH.

Support for No_snoop is system-dependent and, if implemented, No_snoop transforms a final access attribute of a Normal cacheable type to Normal-iNC-oNC-OSH downstream of (or appearing to be performed downstream of) the SMMU. No_snoop does not have an effect on an any-Device type.

Note: This is consistent with SBSA's requirements of No_snoop.

Note: To achieve this 'pull-down' behavior, the No_snoop flag might be carried through the SMMU and used to transform the SMMU output downstream.

13.6.2 ATS attribute overview

It is IMPLEMENTATION DEFINED whether an SMMU implementation assigns attributes (of memory type, Shareability and allocation hints) to ATS "Translated" transactions consistent with the result of an "Untranslated" transaction to the same address, or whether ATS Translated transactions have the fixed cached, shared attribute applied.

Note: An ATS client makes a Translation Request to the SMMU and caches the resulting Physical Address in its local Address Translation Cache (PCI terminology for a remote TLB). When a client has successfully received a translation, it might use it later to make direct physically-addressed ("Translated") accesses to the page, which are intended to bypass SMMU translation altogether. However, the PCIe ATS protocol r1.1 does not explicitly provide a field to convey a memory type/access attribute to the client in the completion of the Translation Request, therefore the subsequent Translated transaction is issued from the endpoint with no attributes (beyond R/W).

When a regular "Untranslated" (non-ATS) transaction is forwarded to the SMMU by the Root Complex, the incoming attribute is constant, fixed at cacheable shareable as above. The SMMU applies an outgoing attribute determined from a function of input, input overrides and TTDs, so the outgoing attribute is address-dependent.

When a Translated transaction (from ATS) is forwarded to the SMMU, it is intended to bypass address translation in the SMMU and, if ATS attributes are supported, the SMMU modifies the transaction's attributes as appropriate for the accessed page.

Note: If ATS attributes are not supported, the resulting output attribute from a Translated transaction might not match that of a non-ATS/Untranslated access to the same address. This is acceptable in systems for which an attribute mismatch does not present a loss of correctness, or where it is expected that the SMMU-assigned attribute will always be consistent with the fixed cached, shared attribute.

If ATS attributes are supported, the mechanism for assigning an attribute to a Translated transaction is implementation specific.

Note: Because the ARM architecture does not support a 64-bit Physical Address space, the MSBs of the 64-bit Translated Address field are ordinarily unused and an implementation might encode attribute information into this space. In compliance with the PCIe ATS architecture [3], a PCIe function with ATS must not modify any Translated Address that is returned in a Translation Completion in any way, and must cache all bits [63:12] of address. If a non-compliant function were to exist that illegally drops some of the Translated Address bits, or otherwise modifies them, it is not able to use ATS with this scheme.

13.6.2.1 Supporting No_snoop with ATS

ARM recommends that, in systems that support No_snoop, it behaves as though the flag passes through the SMMU to downgrade any output cacheable shareable type into non-cached post-SMMU, consistent with SBSA.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If the SMMU supports the encoding of attributes into ATS, the mechanism for determining the ATS Translated attribute in the SMMU is independent of the subsequent application of a No_snoop non-cacheable/downgraded attribute.

The N field in ATS Translation Completions is always 0. The SMMU does not provide means to control No_snoop per-page.

13.6.3 Split-stage (STE.EATS == 0b10) ATS behavior and responses

When Split-stage ATS is enabled for a stream (that is, when STE.EATS == `0b10`, SMMU_CR0.ATSCHK == 1 and STE.Config[2:0] == `0b111`), the response to an ATS Translation Request contains an IPA. When a Translated transaction is emitted as a result of this response, its IPA then undergoes stage 2 translation in the SMMU.

Note: In effect, the stage 1 VA to IPA translation is held in the ATC but stage 2 IPA to PA translation is still performed in the SMMU. This can be used to increase security/isolation over regular nested EATS == `0b01` ATS, where the system does not trust the endpoint to use physical addresses directly.

The behavior of Split-stage ATS is the same as regular EATS == `0b01` ATS with nested (stage 1+stage 2) translation, except:

- The completion contains the IPA result from the Stage 1 translation of the requested VA (instead of the output PA from stage 1+stage 2 translation of the requested VA).
- If the SMMU supports the encoding of attributes into ATS (see section 13.6.2), an attribute is returned that gives a final output equivalent to the Stage 1+Stage 2 combined attribute when the resulting ATS Translated transaction is subject to stage 2 translation in the SMMU.
    - Note: This might be implemented as returning a Stage 1 attribute which is later combined with that of stage 2 in the SMMU, or might be implemented as returning a stage 1+2 combined attribute which might later be passed through stage 2 (or combined with stage 2 a second time, which gives the same result).
- HTTU might be performed by the stage 2 translation that occurs as part of the ATS Translation Request, or it might be performed at the time of the stage 2 translation for a subsequent Translated transaction.

All other aspects are identical to regular EATS == `0b01` ATS with nested translation:

- Permissions are granted from the combination of the stage 1 and stage 2 translation permissions (see 13.7), with respect to the permissions requested in the Translation Request.
- The maximum permissible translation size is derived from the intersection of stage 1 and stage 2 page/block sizes. ARM recommends that, for performance reasons, the maximum translation size is returned where possible.
    - Note: The largest translation size is the minimum of the stage 1 and stage 2 translation sizes for the given address, such that the returned translation represents a single span with constant permission and translation.

See 13.6.5 for behavior of STE.S1DSS skipping stage 1.

The behavior of an incoming Translated transaction differs when the EATS configuration of the StreamID presenting the transaction is `0b10`:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- When EATS == `0b01`, the Translated transaction is presented directly to the output. The given address is treated as a PA. If the SMMU supports the encoding of attributes into ATS, the attribute encoded in the Translated transaction is used as the output attribute, otherwise the (fixed) input attribute is output.
- When EATS == `0b10`, the Translated transaction provides an IPA. This IPA is subject to stage 2 translation (which might generate faults and events) and, if successful, provides the PA for output. If the SMMU supports the encoding of attributes into ATS, the attribute encoded in the Translated transaction is input to stage 2, otherwise the transaction's (fixed) input attribute is input to stage 2. The stage 2 translation's attribute combines with the input to stage 2 to form the output, in the same way as for any other stage 2 translation. The overrides that are configured in STE.INSTCFG and, if SMMU_IDR3.XNX == 1, STE.PRIVCFG, are applied to the Translated transaction prior to stage 2 translation and permission checking.

EATS == `0b10` configuration is only usable when SMMU_CR0.ATSCHK == 1.

Note: ATSCHK == 1 causes Translated transactions to look up and be checked against STE configuration, which for EATS == `0b10` provides the stage 2 with which to translate and for EATS == `0b01` allows the transaction to bypass without translation.

13.6.4 Full ATS skipping stage 1

When all of the following are true, transactions arriving without a PASID are configured to skip stage 1 translation:

- STE.Config[0] == 1 (Stage 1 translation enabled)
- STE.EATS == `0b01` ("Full ATS" enabled)
- STE.S1CDMax != 0 (Substreams/PASIDs enabled)
- STE.S1DSS == `0b01`, (non-PASID transactions bypass Stage 1)

If stage 2 translation is enabled (STE.Config[1] == 1), an ATS Translation Request without a PASID is translated stage 2-only as though STE.Config[1:0] == `0b10`. The Translation Completion returns a PA from the stage 2-only translation.

Where a stage 1 + stage 2 configuration has the first stage skipped due to STE.S1DSS == `0b01`, the translation process behaves as though stage 1 were not configured (corresponding to Config[0] == 0). If ATS attributes are supported, the final attribute is therefore determined by the (fixed) upstream input attribute, replaced by STE overrides where configured, and combined with the attribute from the stage 2 TTD.

If stage 2 is not enabled, an ATS Translation Request without a PASID made to this configuration skips the single translation stage and, if a fault or configuration error has not occurred, its Translation Completion returns a direct-mapped "pass-through" of the stage. Responses to requests that lead to faults and configuration errors are described in section 3.9.1.1.

This is encoded as an identity-mapped Translation Completion containing:

- U == 0
- R == 1
- W == 1
- TranslatedAddress 1:1/identity-mapped with Translation Request's Untranslated Address
- Translation size containing at least the requested Untranslated Address

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

    - Note: An implementation might return a larger identity-mapped region, as any address accessed under the same StreamID/configuration conditions will result in this type of response. A larger region might avoid future page-by-page translation requests.
    - The maximum permissible translation size is the same as the OAS (see section 3.4).
- Note: As this response is made for an ATS Translation Request received without a PASID, the request cannot contain Execute_Requested == 1 or Privileged_Mode_Requested == 1 as these fields are carried in a PASID. The response is made with Execute_Permitted and Privileged_Mode_Access both 0.
- Note: This response causes the function to make 1:1 accesses to physical addresses, equivalent to the behavior of untranslated non-ATS transactions from the same stream.

The output attribute is a function of the (fixed) upstream input attribute replaced by STE overrides (where configured); this can be evaluated at the time of the Translation Request and encoded into the returned Translated Address as per section 13.6.2.

13.6.5 Split-stage ATS skipping stage 1

Where

- STE.Config[1:0] == `0b11` (Stage 1 and 2 translation enabled)
- STE.EATS == `0b10` (Split-stage ATS enabled)
- STE.S1CDMax > 0 (Substreams/PASIDs enabled)
- STE.S1DSS == `0b01`, (non-PASID transactions bypass stage 1)

a Translation Request without a PASID is configured to skip stage 1 translation, but the ATS configuration is for stage 1-only.

The Translation Request might lead to a fault or configuration error that triggers an ATS response as described in section 3.9.1.1.

Note: Even if stage 1 is effectively bypassed, a stage 1 Address Size Fault might occur if the input address supplied in the Translation Request is greater than the IAS.

Note: A Permission fault leads to one or more of R, W and X permissions being denied in the response, as shown in the next paragraph.

Where a Translation Request with a Split-stage ATS configuration skips stage 1, the Translation Request supplies an IPA address and, if a fault or configuration error does not occur, an identity-mapped Translation Completion is returned, which contains:

- U == 0
- R and W permissions are granted from the stage 2 permissions for the IPA, with respect to the permissions requested in the Translation Request.
- TranslatedAddress 1:1/identity-mapped with the Translation Request's IPA.
- The maximum permissible translation size is that of the stage 2 translation. ARM recommends that, for performance reasons, the maximum translation size is returned where possible.
- Note: See 13.6.4; this response is made with Execute_Permitted and Privileged_Mode_Access both 0 as it is in response to a request made without a PASID.

If the SMMU supports the encoding of attributes into ATS (see 13.6.2), an attribute is returned that causes the resulting ATS Translated transaction to have a final output (after Stage 2 translation in the SMMU) equivalent to the (fixed) upstream input attribute, replaced by STE overrides (where configured), combined with the Stage 2 translation's attribute.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: This might be implemented by returning the constant input attribute (cached, shareable) replaced by STE overrides, and later combining that during stage 2 translation. Or, this might be implemented by returning the input (with STE overrides) combined with stage 2 translation and either passing the Translated transaction's attribute through or combining it with stage 2 translation again (which gives the same result).

The function then performs Translated accesses with IPAs and these are translated stage 2-only in the same way as described above for Split-stage ATS that does not skip stage 1.

13.7 PCIe permission attribute interpretation

PCIe-domain permissions are interpreted by the SMMU as described in this section.

Base PCIe interconnect expresses only a Read/Write access permission. The PASID TLP prefix adds the following access permissions:

- Execute (In the PASID, Execute_Requested, hereafter referred to as 'Exe')
- Privileged (In the PASID, Privileged_Mode_Requested, hereafter referred to as 'Priv')

For normal transactions, the SMMU can use R/W and Privileged directly as incoming R/W and PRIV attributes.

Note: The PASID TLP prefix can only encode an 'Execute' attribute for Memory Read Requests; this is compatible with the SMMU's behavior in considering all write transactions to be Data.

The SMMU interprets the INST and PRIV attributes of a normal transaction without a PASID TLP prefix as "Data" and "non-privileged", respectively.

For an ATS Translation Request, base PCIe supplies:

- Read access is implied in all ATS Translation Requests
    - An ATS Translation Completion might grant per-page Read access, depending on page permissions.
- No-Write (NW)
    - NW == 0 signals the intention of the device to perform write accesses. When NW == 0, the Translation Completion sent by the SMMU grants Write permission if the page permission (from TTD.AP and/or TTD.HAP) contains permission for write (Note: pages marked Dirty have write permission). When NW == 1, Write permission is permitted but not required to be granted if the page permissions contain permission for write.
    - When HTTU is not enabled, Write permission is not granted when the page permissions do not contain permission for write regardless of the request's NW bit or the TTD DBM state.
    - When HTTU is enabled, a request having NW == 0 to a page marked Writable Clean updates the page to be marked Dirty and then grants Write permission in the response. See 3.13.6.

      Note: Referring to the above, a request with NW == 1 might grant write access if the page is already marked Dirty (as the page permissions contain write permission).
    - HTTU does not mark a page Dirty in response to a Translation Request with NW == 1.

The PASID TLP prefix adds the following to an ATS Translation Request:

- Execute (Exe)
    - This flag requests execute permission. If this flag is set, the response might grant execute permission. The response must not grant execute permission if this flag is not set in the request.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

A valid translation might or might not grant the requested Execute permission, depending on actual page 'X' permission.

Note: A request might be made with Exe == 1 and NW == 0, meaning an endpoint requests access for write and execute. This does not imply that the endpoint will perform 'instruction writes' to the page.

- Exe implies R in an ATS response.

  Certain configurations of AArch64 translation tables allow an 'XO' execute-only permission. A translation that requests Exe permission through ATS is not compatible with an XO page and will result in no access, unless STE.INSTCFG == Instruction. Similarly, care must be taken when using the Stage 2 'XO' permission with ATS; a guest might map all executable pages as RX but a hypervisor Stage 2 can further modify this down to execute-only. See section 13.7.1 below for more information on ATS Exe permission.

- Privileged (Priv)
  - This flag marks the request as being made from a privileged or unprivileged entity in the endpoint. The response contains permissions granted only to that entity/privilege level.

    Note: The PCIe PASID ECN [9] states: "The ATC must not assume any correlation between the Privileged Mode and Non-Privileged Mode permissions associated with a translation."

For an ATS Translation Request without a PASID TLP prefix, the INST and PRIV attributes must be provided to the SMMU as "Data" and "non-privileged", respectively.

The following table of example ATS Translation Requests shows the access permissions granted for the request on several different page permission combinations:

Example Request	Page permissions	Response
NW == 1, Exe == 0, Priv == 0	User-RO, Priv-RW, XN == 0	R == 1, W == 0, Exe == 0, Priv == 0
NW == 0, Exe == 0, Priv == 0	User-RW, Priv-RW, XN == 0	R == 1, W == 1, Exe == 0, Priv == 0
NW == 0, Exe == 0, Priv == 0	User-RO, Priv-RW, XN == 0	R == 1, W == 0, Exe == 0, Priv == 0
NW == 0, Exe == 0, Priv == 1	User-RO, Priv-RW, XN == 0	R == 1, W == 1, Exe == 0, Priv == 1
NW == 1, Exe == 1, Priv == 0	User-RW, Priv-RW, XN == 1	R == 1, W == 0 or 1, Exe == 0, Priv == 0 Note: The SMMU is permitted to return Write permission if the page is writable, even if NW == 1.
NW == 0, Exe == 0, Priv == 0	User-RW, Priv-RW, XN == 1	R == 1, W == 1, Exe == 0, Priv == 0
NW == 0, Exe == 1, Priv == 0	User-RW, Priv-RW, XN == 0	R == 1, W == 1, Exe == 1, Priv == 0
NW == 0, Exe == 1, Priv == 0	User-X, Priv-RW, UXN == 0 (AArch64)	R == 0, W == 0, Exe == 0, Priv == 0
NW == any, Exe == any, Priv == any	Not mapped (or any type of translation-related fault)	R == 0, W == 0, Exe == 0, Priv == 0 Note: See 3.9.1; this Translation Completion is marked with 'Success' status, as opposed to UR/CA.

When the STE.INSTCFG and STE.PRIVCFG override fields are supported (SMMU_IDR1.ATTR_PERMS_OVR == 1), they affect ATS Translation Requests as described in 13.7.1 below.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: ARM recommends that an STE not override INST and PRIV when the system supports PCIe PASID prefixes, in order for this attribute to be communicated from the device to the SMMU translation lookup. However, when a PASID TLP prefix is not used and the INST and PRIV attributes are provided as “Data” and “non-privileged”, as required above, software might wish to override these attributes using the STE input overrides; for example, it might configure a stream to be privileged. When PASIDs/SubstreamIDs are configured for a translating Stage 1 configuration, traffic without a PASID is deemed unexpected if STE.S1DSS == `0b00`, and aborted. If STE.S1DSS == `0b01`, traffic without a PASID skips Stage 1 and translation behaves as though the STE is configured for Stage 2-only translation. In this case, the PRIV attribute is ignored by Stage 2 and the “Data” default is sensible.

When HTTU is in use, an ATS TR that generates a response with R == 1 || W == 1 || Exe == 1 also sets the TTD.AF == 1. When HTTU is enabled for Dirty state update (using HD flags), an ATS TR having NW == 0 to a Writable-Clean page accessible at the requested permission level (permissions are read-only, with DBM == 1) will make the page permissions read/write (Writable-Dirty) and generates a response with W == 1. An ATS TR having NW == 1 must not mark a page as Writable-Dirty. (Note: TTW in nested stage 1+stage 2 scenarios might cause stage 2 pages to be marked Writable-Dirty where Stage 1 TTDs are updated)

Note: As STE.{INSTCFG,PRIVCFG} overrides might affect the permissions granted in an ATS response, a change to these overrides must be accompanied by an invalidation of associated ATCs.

Note: When Split-stage ATS is in use, stage 2 translation is performed on incoming Translated transactions. A Translated transaction is presented from PCIe as Data, Unprivileged. This means that it is possible to configure stage 2 execute permissions so that the ATS translation request succeeds, but the subsequent Translated transaction is not permitted. For example:

1. A Split-stage ATS Translation Request from a stream with STE.INSTCFG == Instruction is made to an address where stage 1 has RWX permissions and stage 2 has Privileged-XO permissions.
2. The ATS Translation Request succeeds because the combined stage 1+2 permission is Privileged-XO and the STE.INSTCFG configuration treats all accesses as execute.
3. The Translated transaction is translated at stage 2, but is presented to the SMMU as Unprivileged, which causes a stage 2 permission fault.

Note: Use of Split-stage ATS with STE.INSTCFG, or stage 2 permissions that are not expressible in PCIe, can lead to unexpected results. ARM recommends that STE.INSTCFG is not used with ATS.

13.7.1 Permission attributes granted in ATS Translation Completions

The pseudocode below illustrates how the R, W, Exec, Priv attributes in a Translation Completion are calculated with respect to the input Translation Request, the translation table descriptor permissions and the STE INSTCFG/PRIVCFG overrides.

A PRIVCFG == Privileged or PRIVCFG == Unprivileged override calculates (R,W,X) attributes appropriate to the privilege level to which the request was overridden, but the ‘Priv’ field in the ATS Translation Completion returns the same privilege supplied in the associated Translation Request.

- For example, a TR with ‘NW == 0, Exe == 0, Priv == 1’ with PRIVCFG == Unprivileged, and page permissions of ‘User-RO, Priv-RW’, returns a Translation Completion with ‘R == 1, W == 0, Exe == 0, Priv == 1’.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The INSTCFG override affects the interpretation of the TR's Exe and the translation table 'X' attributes.

```
// Here, TR is the incoming Translation Request, TC is the returned
// Translation Completion and final_combined_translation is the
// result of all enabled stages of translation for the given address.
// HTTU is not shown; it is assumed final_combined_translation contains
// post-HTTU permissions if relevant.  (Note:  HTTU for ATS is influenced
// by TR.NW.)

if (!TR.PASID_present) {
      // The Priv and Exe fields of a Translation Completion without a
      // PASID are reserved and set to 0.  Effectively, there are no Priv
      // and Exe permission bits in a response without a PASID:
      TR.Priv = 0;
      TR.Exe = 0;
}

// The Translation Completion's Priv field is not affected by PRIVCFG and
// is as the TR supplied:
TC.Priv = TR.Priv;

STE_effective_PRIVCFG = (SMMU_IDR1.ATTR_PERMS_OVR == 1) ? STE.PRIVCFG :
                        Use_Incoming;
STE_effective_INSTCFG = (SMMU_IDR1.ATTR_PERMS_OVR == 1) ? STE.INSTCFG :
                        Use_Incoming;

// Effective privilege that permissions will be checked against:
effectivePriv = (STE_effective_PRIVCFG == Use_Incoming) ? TR.Priv :
                  ((STE_effective_PRIVCFG == PRIVILEGED) ? 1 : 0));

ttd_R = final_combined_translation.readable_by_privlevel(effectivePriv);
ttd_RX = ttd_R &&
      final_combined_translation.executable_by_privlevel(effectivePriv);
ttd_X = final_combined_translation.executable_by_privlevel(effectivePriv);

// Grant of write permission is always governed by the translation's W
// permission.  Also note that, when enabled, HTTU only sets Dirty if
// TR.NW == 0; the page is writable if this is the case, or if the page is
// already Dirty (see text).
// Note:  An implementation is not *required* to return W == 1 if
// TR.NW == 1 and TTD permissions include write, but doing so can avoid
// the Endpoint having to make a second request if it subsequently requires
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

```
// to write.
TC.W = final_combined_translation.writable_by_privlevel(effectivePriv);

if (STE_effective_INSTCFG == Use_Incoming) {
      TC.R = ttd_R;
      TC.Exe = TR.Exe && ttd_RX;    // Granting X implies granting R
      // Note:  An execute-only (XO) translation grants no access to ATS.
      //        A normal (non-ATS) transaction for Exe == 1 will succeed
      //        to an XO translation.
} else if (STE_effective_INSTCFG == INST) {
      TC.R = ttd_X;
      TC.Exe = TR.Exe && ttd_X;
      // Note:  An XO translation can grant R&Exe access to an ATS Request
      //        because in this configuration all reads are considered
      //        instruction fetches, therefore R&Exe are the same.
} else { // STE.INSTCFG == DATA
      // This is effectively equivalent to assuming X=R; anything with ‘R’
      // for the privilege level is accessible.
      TC.R = ttd_R;
      TC.Exe = TR.Exe && ttd_R;
}
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

14 External interfaces

14.1 Data path ingress/egress ports

Along with read/write data/address, the following information is supplied with an incoming transaction:

- StreamID
- Optionally, SubstreamID (PASID) and a SubstreamID-Valid flag, SSV.
- StreamID is qualified by a SEC_SID flag to denote Secure/Non-secure StreamID when the SMMU implements security support.
- StreamID is passed through the SMMU into the memory system, to create a DeviceID that enables the GICv3 ITS to differentiate interrupts by stream.
    - An internal StreamID is generated for MSIs originating from the SMMU.
        - Note: The StreamID generated for MSIs must have a different value to those associated with client devices, so that the GICv3 ITS can differentiate SMMU MSIs from those originating from client devices.
- ATS Translated/Untranslated tag to control bypass.
- Instruction/Data/NS permission attributes and Memory type/Shareability/allocation hints from upstream device (optional, can be overridden in STE). See section 13.

If the ability to perform HTTU atomic updates using local monitors is required, the SMMU would need to attach to the system using a fully-coherent interconnect port. However, if HTTU is not implemented or the downstream system provides far atomic facilities which do not require a fully-coherent port, an IO-coherent interconnect port might be used.

As the SMMU does not translate outgoing coherency or broadcast invalidation traffic, there is no requirement to use an interconnect supporting cache coherency or DVM between the SMMU and client devices. Client devices might connect to the SMMU using an IO-coherent interconnect port.

14.2 ATS Interface, packets, protocol

Note: An SMMU implementation might provide a separate interface to provide ATS and PRI protocol support with a compatible PCIe Root Complex. This interface is outside the scope of this specification.

14.3 SMMU-originated transactions

An SMMU read for any translation, configuration or queue structure that is performed into any PCIe address space is permitted to return any value or be terminated with an external abort.

Note: ARM expects SMMU structures and translation tables that are accessed externally in non-embedded implementations to be located in system memory. A potential deadlock (where an SMMU read access is dependent on the completion of an incoming PCIe write which is itself dependent on the SMMU translation that caused the original access) can be avoided by the system terminating SMMU accesses targeted to the PCIe domain by malicious or incorrect software.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

15 Translation procedure

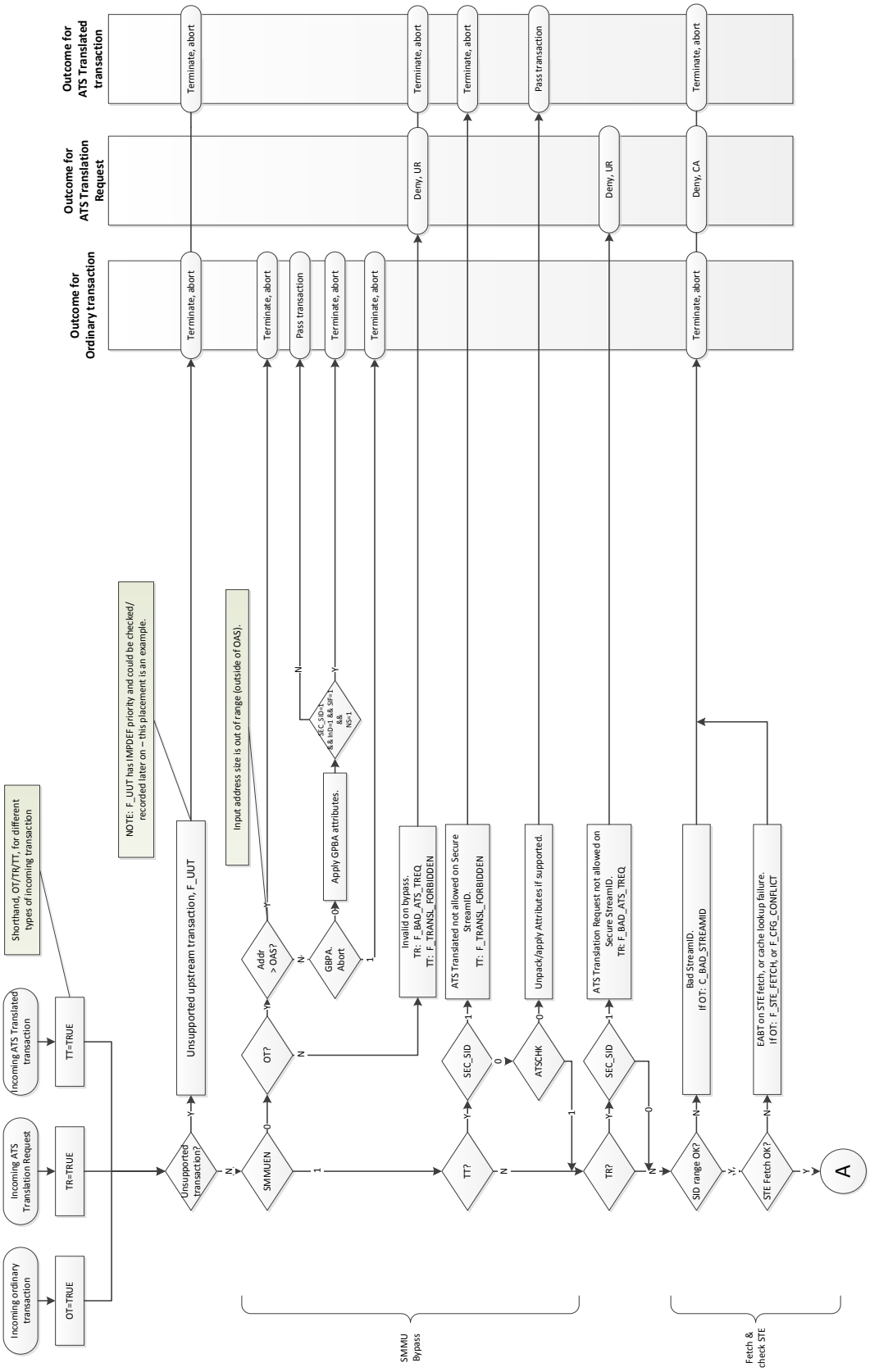
The following flowcharts are an illustration of the sequence of events from the beginning of a translation to its final outcome. They represent an abstracted translation flow to summarize the information in the rest of this document.

The purpose is to indicate the end result of different types of transactions, in terms of transaction and translation success or error responses (including PCIe ATS errors and completion responses). Certain aspects are not intended to be depicted in detail, including but not limited to:

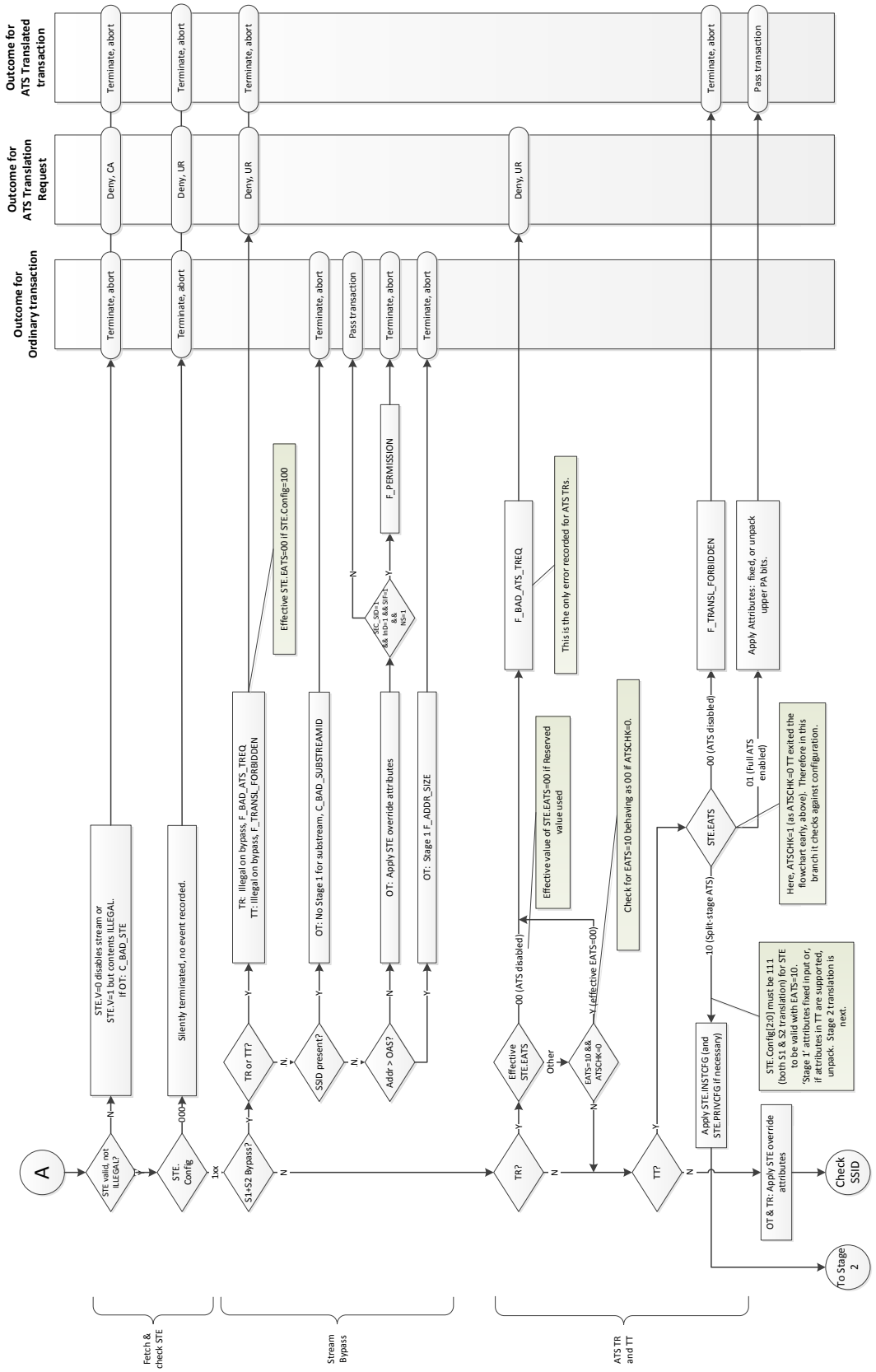
- Atomic translation table update mechanism.
- TLB conflict, configuration cache conflict (which might happen at an IMPLEMENTATION DEFINED point in the translation).
- Speculative operations (which do not record errors or faults).
- Attribute control.

15.1 Translation procedure charts

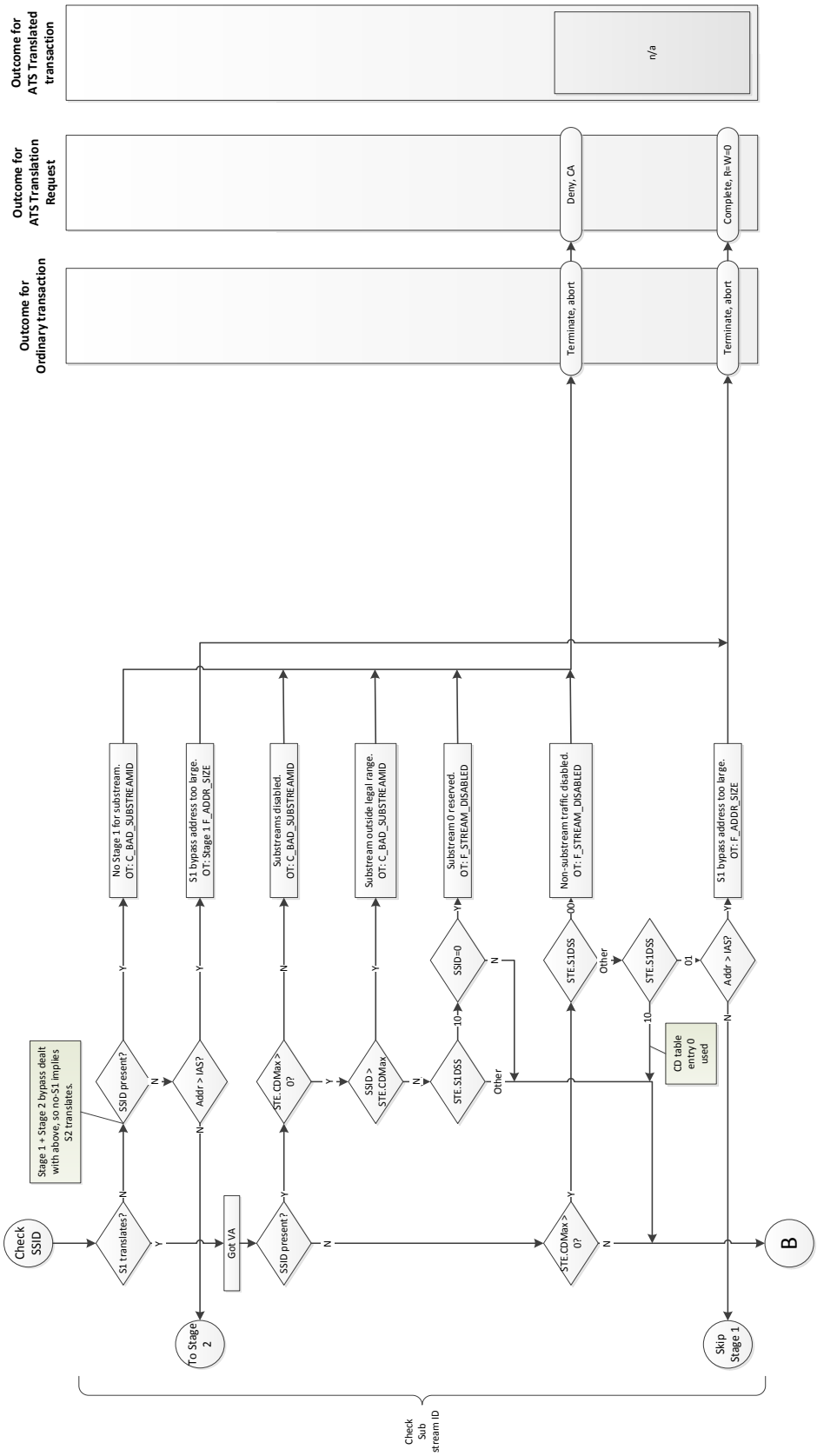
Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential



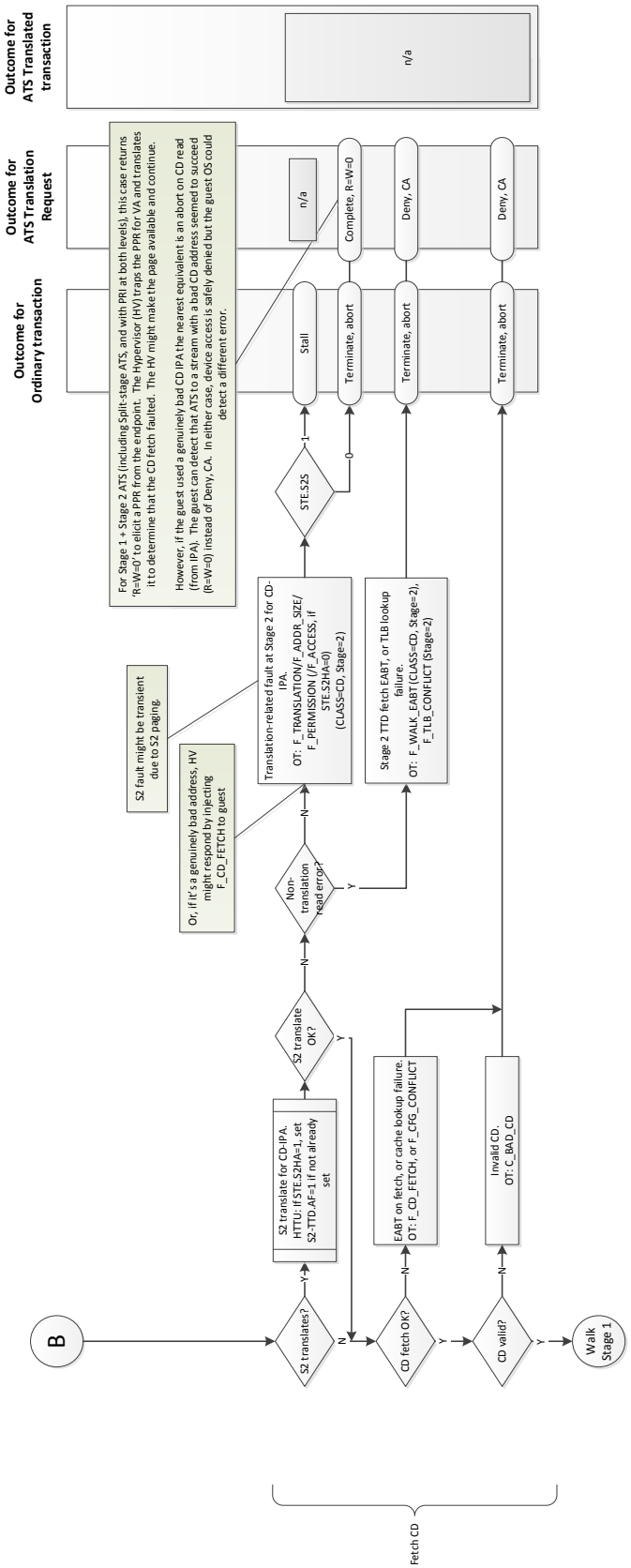
Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential



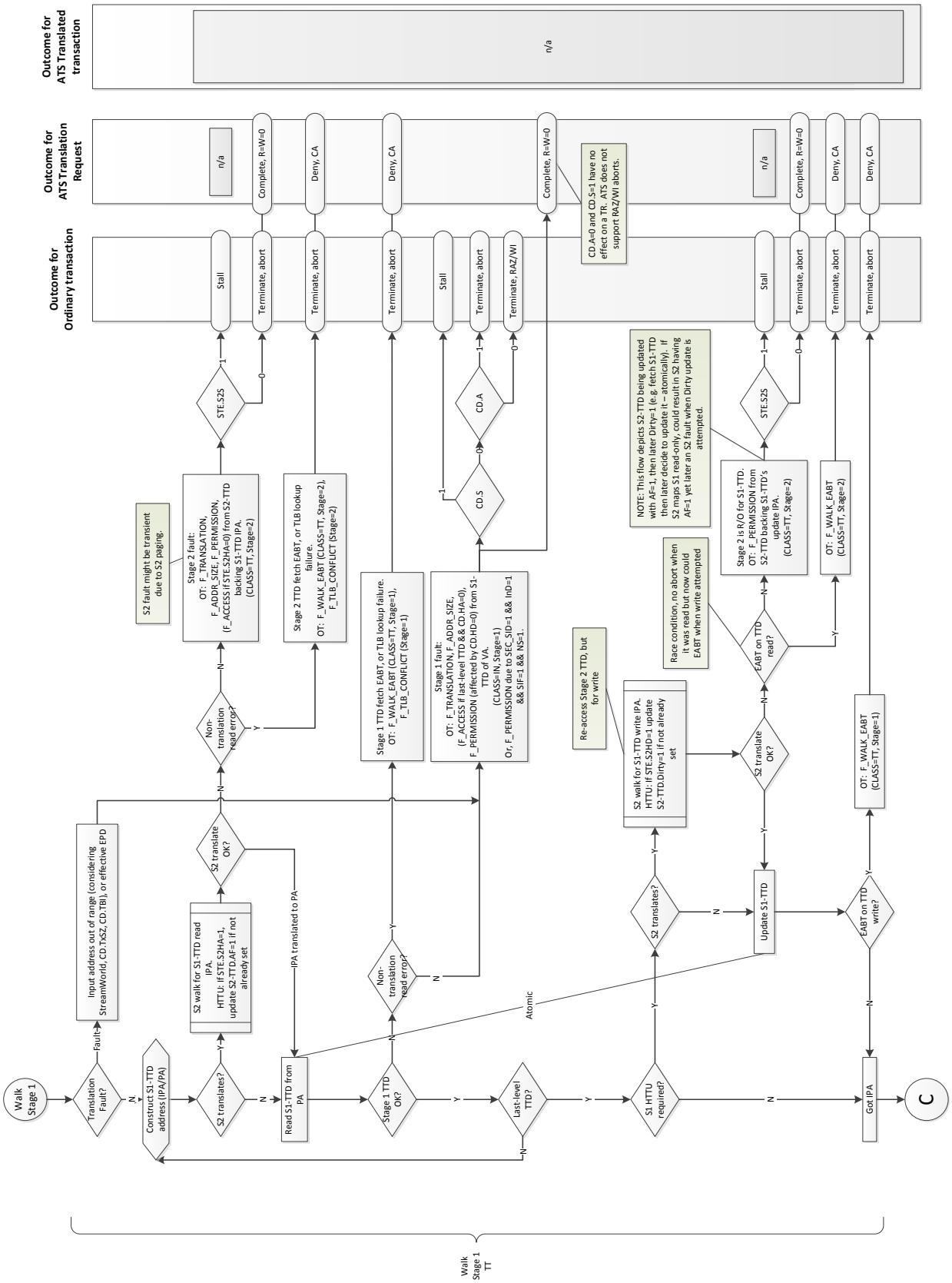
Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential



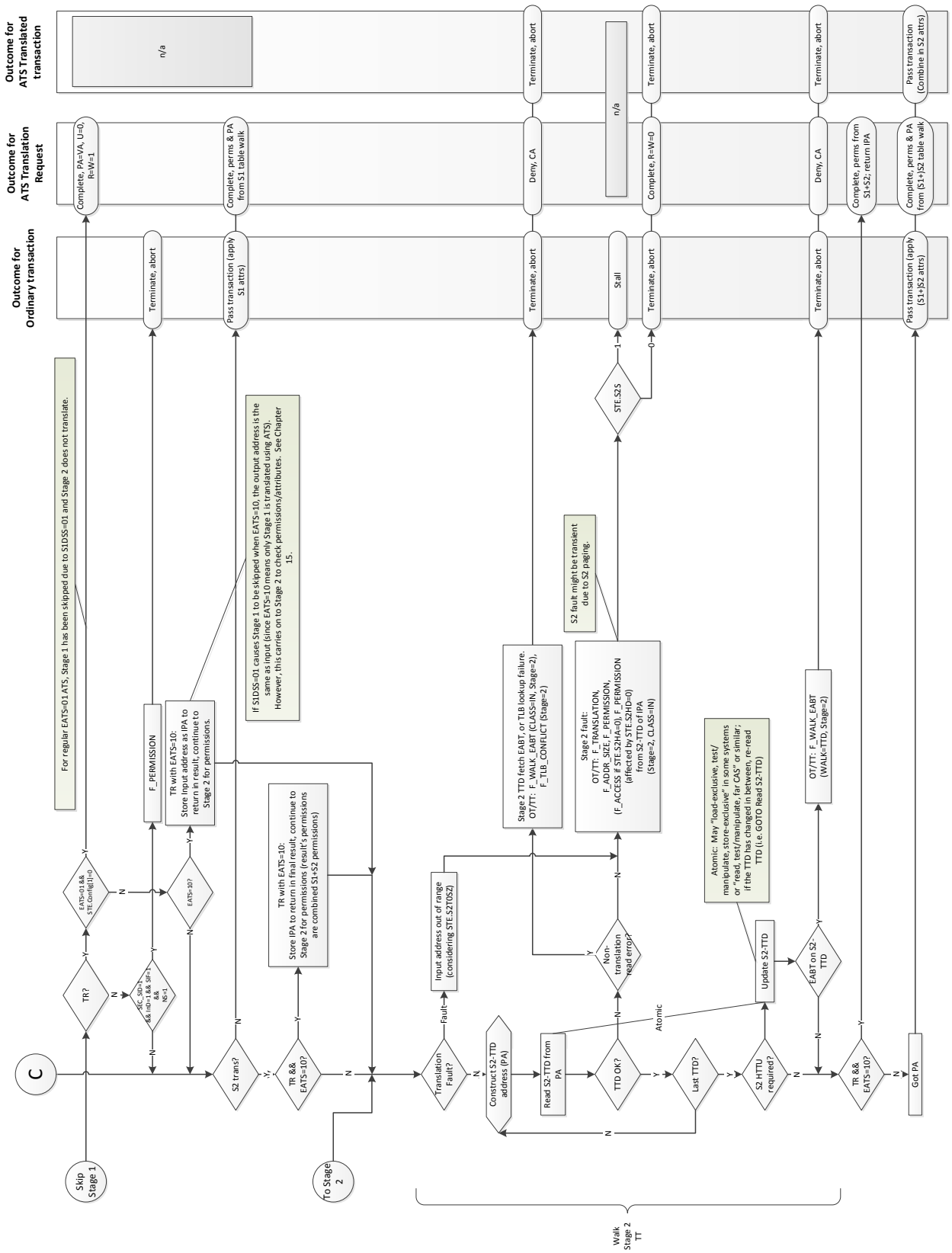
Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential



Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential



Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential



Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

15.2 Notes on translation procedure charts

In general, ATS Translation Requests do not record faults and errors (with the exception of F_BAD_ATS_TREQ which indicates situations that would not otherwise be observable through error responses for an ordinary transaction sent through identical configuration).

In general, ATS Translated traffic does not record faults and errors, with the exceptions of:

- F_TRANSL_FORBIDDEN.
- EATS == `0b10` Split-stage ATS configurations, which are translated at stage 2 in an identical way to ordinary transactions (including recording of faults).

For every fault or termination that an ordinary transaction might experience, an ATS Translation Request has an equivalent defined response. Generally, situations that represent a configuration error result in a Completer Abort (CA) response to the endpoint, situations that represent an explicit prevention or disable of ATS service result in an Unsupported Request (UR) response, and Translation-related failures result in a successful Translation Completion having R == W == 0 (that is, no access for this address).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

16 System and Implementation Considerations

16.1 Stages

ARM strongly recommends that stage 2 is only used to provide device assignment to a guest OS. To support other usage scenarios, ARM recommends implementations also implement stage 1 if stage 2 is implemented.

16.2 Caching

An SMMU implementation is not required to implement caching of any kind, but ARM expects that performance requirements will require caching of at least some configuration or translation information.

Caching of configuration or translations might manifest as separate caches for each type of structure, or some combination of structures into a smaller number of caches. When architected structure types are cached together as one combined entry, the invalidation and lookup semantics remain identical to many specialised per-structure caches.

For example, an implementation with caches of each structure and translation stage implemented separately would contain:

- An STE cache
  - Indexed by StreamID.
  - Invalidated by StreamID, StreamID-span, or all.
  - Might contain a Level 1 Stream table cache of pointers to 2nd-level Stream table.
    - Identical index/invalidation requirements.
- A CD cache
  - Indexed by StreamID and SubstreamID, or address.
    - Note: Address index could be used, calculated from the STE.S1CtxPtr of a prior STE lookup plus the incoming SubstreamID. This arrangement might be useful where many STEs share a common table of CDs.
  - Invalidated by StreamID and SubstreamID, StreamID, or all.
  - Might contain a Level 1 CD table cache of pointers to 2nd-level CD table.
    - Identical index and invalidation requirements.
- A stage 1 TLB (VA to IPA).
  - Indexed by VA, ASID, VMID and EL.
    - EL is the Exception level or StreamWorld on whose behalf the TLB entry was inserted.
  - Invalidated by VA, ASID, VMID and EL, or ASID, VMID and EL, or VMID and EL, or EL.
  - Might contain or be paired with a walk cache (invalidated under the same conditions as PE translation walk caches).
- A stage 2 TLB (IPA to PA)
  - Indexed by IPA and VMID.
    - No Exception level, as stage 2 is only relevant to one StreamWorld (NS-EL1).
  - Invalidated by IPA and VMID, or VMID, or all.
  - Might contain or be paired with a walk cache.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

In this example, consistency is maintained by looking up cache entries in order from STE through CD, then using the parameters determined from that stream configuration to lookup in the TLBs. That is, given a StreamID, SubstreamID and VA input, convert StreamID and SubstreamID into ASID, VMID and Exception level, then use VA, ASID, VMID and EL to lookup in the TLBs.

16.2.1 Caching combined structures

Note: In the example given in 16.2, a stage 1 TLB is implemented separately to the stage 2 TLB. While this layout mimics the two stages of translation table, it might not provide optimal performance. A designer might determine that a combined stage 1 and stage 2 TLB is better, where each entry would translate VA to PA directly but would inherit invalidation requirements from both original structures.

For translations and TLBs, the SMMU invalidation rules for combined TLBs directly match those of ARMv8-A.

Note: For example, an invalidation by IPA is not required to invalidate entries in a combined S1 and S2 TLB but is required to be paired with a second invalidation by VA or stage 1-all that would affect S1 and S2 TLB entries.

Combined structure caches maintain the same invalidation semantics as discrete structures. An entry of a combined cache is invalidated if any part of the entry would have been invalidated in an equivalent operation, with the same parameters, on a discrete cache.

Note: For example, a particular SMMU implementation maintains only two hardware caches, a combined cache of STE and CD, and a translation cache or TLB. In this layout, a StreamID and SubstreamID and VA input looks up StreamID and SubstreamID in the combined cache and determines the ASID, VMID, and StreamWorld at the same time. The translation is then looked up using VA, ASID, VMID and StreamWorld to determine the PA. The invalidation requirements of the combined STE and CD cache are the union of the requirements of the separate structures. STE invalidation operations invalidate every combined cache entry that contains data loaded using a given StreamID. This covers all CDs fetched from a given STE, which is implied anyway by the CMD_CFGI_STE to invalidate CDs subordinate to the STE. Conversely, every entry that contains data from a CD to be invalidated must be invalidated even if the STE portion is still valid.

An implementation might combine a TLB with configuration caching so that a single cache is looked up by StreamID and SubstreamID and VA and results in a PA output. Entries in this cache are invalidated when any part of an entry would match (or cannot be proven to not match) a required invalidation for STE, CD or translation.

Note: Implementations must balance a trade-off between over-invalidation that might be necessary to cover all required entries, and the cost of adding extra tagging. For example, a single cache might tag entries by VA, ASID, VMID, and StreamID so that broadcast TLB invalidations can remove only relevant entries, or so that an STE invalidation removes only entries that could have been constructed from the given StreamID.

16.2.2 Data dependencies between structures

The configuration structures logically make a tree or graph by indicating subsequent structures (and onwards, indicating translation tables). The structures contain fields to locate the next structure in the chain but might also modify interpretation of subsequent structures.

The dependencies between structures are:

- STE to CD to TT (stage 1)

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- STE to TT (stage 2)

(Here, 'STE' might be composed of multi-level L1STD to STE lookups and CD might be composed of multi-level L1CD to CD lookups.)

The STE contains fields that determine how to locate a CD and stage 2 translations (whichever are relevant to STE.Config) but also contains fields that modify the behavior of a CD and translation table walks performed through it:

- The STE StreamWorld (STRW plus STE Secure/Non-secure state) determines the translation regime, which:
    - Tags caches of translations subsequently inserted, to separate lookup and match on invalidation
    - Determines which translation table formats are valid for use by the CD.
- STE.S1STALLD modifies CD.S behavior upon stage 1 fault.
- AArch32 stage 2 translation (STE.Config[1]== 1 and STE.S2AA64 == 0) causes a 64-bit stage 1 CD (CD.AA64 == 1) to be ILLEGAL.

Note: A change to an STE field requires an STE invalidation. An STE invalidation also invalidates all CDs that were cached through the STE.

The CD contains fields that determine how to locate translation tables but also contains fields that modify the behavior of a translation table walk through it:

- CD.{AA64, EPDx, SHx, ORx, IRx, TGx, TxSZ, ENDI, NSCFGx} govern walks of the translation table itself. In addition, NSCFGx can influence the final NS attribute because a Non-secure TTW never outputs NS == 0.
- CD.{UWXN, WXN, PAN, AFFD, HADx} govern permission checking with the translation table descriptors.
- CD.{ASID, ASET} govern ASID-tagged translation cache entries.
- CD.{MAIR, AMAIR} modify the attribute determined from translation.
- CD.{HA,HD} determines HTTU configuration for walks performed through TTB{0,1}.

Some STE and CD fields are permitted to be cached as part of a translation or TLB entry (therefore requiring invalidation of TLB entries that might contain the old value when the fields are changed). These fields are noted in sections 5.2 and 5.4.1. With the exception of these fields, no other information is expected to be 'carried forward' between structures.

Some configuration register fields are, where indicated, permitted to be cached in a TLB. Changes to these fields require invalidation of any TLB entries that might cache a previous value of the field.

16.3 Programming implications of bus address sizing

If pointers are programmed into a device from the PE that would, on the PE, cause translation faults due to failing sign-extension checks, the SMMU will also raise a translation fault because of the sign-extension checks on input. However, if a system cannot convey all 64 address bits from a device, or a device lacks the ability to register upper address bits, the SMMU does not have enough information to perform these checks. In this case, ARM recommends that if detection of such errors is required, software (or the device, if it has the facility to hold the full address) checks the validity of upper bits.

On ARMv8-A PEs, the TBI facility allows the top byte of addresses to contain tags that are ignored when checking address sign-extension validity. If an address is truncated to ≤56 bits on the flow through device DMA registers to

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

device DMA accesses to I/O interconnect to the SMMU, the SMMU cannot check the validity of the top byte so effectively TBI is always used. In such a system, another entity (device, software) must check addresses if required.

16.4 System integration

- The SMMU must be in the same Inner Shareable domain as any other agents that might use DVM with the SMMU, because DVM messages are only broadcast over the Inner Shareable domain.
- In general, ARM does not expect SMMUs to be connected in series.

  Note: This topology needs special software support, particularly when different software modules manage different SMMUs. This must not be used to construct two stages of translation using two SMMU implementations that support only one stage, as it is programmed differently to an SMMU that supports both stages.
- When used with a PCIe subsystem, an SMMU implementation must support at least the full (16-bit) range of PCI RequesterIDs and the system must ensure that a Root Complex generates StreamIDs from PCI RequesterIDs (BDF) in a one to one or linear fashion so that StreamID[15:0] == RequesterID[15:0]. A larger StreamID might be constructed by concatenating the RequesterIDs from multiple PCI domains (or "segments" in ACPI terminology), for example:
  - StreamID[17:0] == { pci_rc_id[1:0], pci_bus[7:0], pci_dev[4:0], pci_fn[2:0] };

    that is, StreamID[17:0] == { pci_domain[1:0], RequesterID[15:0] };

When used with a PCIe system supporting PASIDs, ARM recommends that the SMMU supports the same number of (or fewer) PASID bits supported by client Root Complexes so that software is able to detect end-to-end SubstreamID capabilities through the SMMU.

- If accesses from a device are expected to experience page faults and the Stall model is used, ARM recommends that a system does not depend on other devices on the same SMMU path as the device in order to resolve the faults. Because a stalled transaction occupies an input buffer resource, the SMMU might not guarantee to pass traffic whether faulting or not, and any new request for device DMA might deadlock.
- Streams belonging to PCIe endpoints must not be stalled. The Terminate model is the only useful option. Stalling PCIe transactions risks either timeouts from the PCIe endpoint (which might be difficult to recover from), or deadlock in certain scenarios.
- Specifically, PCIe traffic (especially if configured to Terminate, architecturally not stalling) must not be held up waiting for any PE action, including draining the Event queue or restarting stalled transactions. PCIe traffic must always make forward progress without unbounded delays dependent on software. An implementation must ensure that transactions to be terminated are not blocked by any other users of the SMMU which might consume resources or stall transactions for an indefinite time.

16.5 System software

Note: Software must:

- Not assume that both stage 1 and stage 2 are implemented.
- Support systems in which broadcast TLB invalidation messages are not supported so do not invalidate SMMU TLB entries, that is fall back to software TLB invalidation messages.
- Discover StreamID and SubstreamID sizes and capabilities.
- Probe SMMU_IDR1 for PRESET configuration table and queue base pointers, only allocating memory for pointers that require initialization.
- Discover the maximum table sizes of the SMMU rather than using fixed-size tables.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Interoperate between Secure and Non-secure domains and function whether or not the other Security state exists. (Implementations might be NS, Secure-only or of mixed security.)
- Present system-specific StreamIDs as part of firmware descriptions for each device, as the StreamIDs associated with a physical device are system-specific.
- Ensure that when HTTU is not used, TTDs mapping DMA memory are marked Accessed (and not Read-Only, if DMA writes are expected) in order to avoid faults.

16.6 Implementation-defined features

16.6.1 Configuration and translation cache locking

Note: The lockdown of configuration and translation cache entries is not a feature directly described by the SMMU architecture because cache structures might vary between implementation and entry lockdown might expose this layout to software.

An implementation might support cache locking in an IMPLEMENTATION DEFINED manner using registers in the IMPLEMENTATION DEFINED memory map.

Note: These registers might expose cache contents and provide insertion, probe and invalidation operations.

If an implementation supports translation cache locking, TLB invalidation must be consistent with the ARMv8-A rules on locked entries:

- A TLB invalidate-all operation does not invalidate locked entries.
- An implementation might choose to implement TLB invalidate-by-VA or invalidate-by-ASID operations so that they do one of the following:
    - Invalidate locked entries that are explicitly matched by the operation.
    - Do not invalidate locked entries.

A locked TLB entry is not affected by over-invalidation side-effects of invalidation operations that do not directly match the entry.

The behavior of CMD_CFGI_* commands with respect to locked configuration cache entries is IMPLEMENTATION DEFINED.

See SMMU_S_INIT.INV_ALL in section 6.3.53, this initialization invalidate-all operation invalidates locked entries.

16.7 Interconnect-specific features

16.7.1 Reporting of Unsupported Client Transactions

The SMMU behaves as though a single transaction is associated with one translation.

SMMU implementations might define their own input alignment restrictions leading to an unsupported client transaction error. For example, an implementation with an AMBA downstream interconnect is likely to treat an incoming transaction that crosses a 4KB boundary as unsupported, because these would violate the alignment rules of the downstream interconnect.

For AMBA4 systems the following upstream client transactions are unsupported:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Far Atomic operations (see section 16.7.6) where not supported by the downstream interconnect or SMMU implementation.

Such transactions will be aborted and an F_UUT event will be recorded if possible.

16.7.2 Non-data transfer transactions

Some interconnect architectures support transactions that do not perform a data transfer, prefetch or translation request action.

If the input interconnect can express the following transactions from client devices, the transactions will be terminated silently by the SMMU with a Slave Error (or equivalent on non-AMBA interconnect):

- DVM operations (of all sub-types).
- Barriers.

The interconnect architecture of an implementation might support the following non-data operations that perform address-based cache maintenance:

- Clean.
- Invalidate.
- CleanInvalidate.
- CleanToPersistence.
    - In AMBA, these are the MakeClean, MakeInvalid, CleanInvalid and CleanShared (with point of persistence scope) transactions.

SMMUv3.0:

- The SMMU does not support Cache Maintenance Operations and silently terminates them.

SMMUv3.1:

- Cache Maintenance Operations that are not address-based are not supported and are silently terminated by the SMMU.
- Cache Maintenance Operations are permitted to pass into the system, without the transformation described in section 16.7.2.1, when the transaction bypasses all implemented stages of translation. See section 16.7.2.3 on memory types.
    - Note: For a Secure stream, SMMU_S_IDR0.SIF still applies. See section 16.7.2.2.
- When one or more stages of translation is applied, the SMMU allows these operations to progress into the system subject to the configuration controls and permission model described in sections 16.7.2.1 and 16.7.2.2. Additionally, Invalidate operations might be transformed into Clean and Invalidate operations as part of these checks.
- When the input interconnect can deliver these operations, but the output interconnect does not support them, the transactions are silently terminated.

SMMU implementations supporting AMBA might define an IMPLEMENTATION DEFINED set of unsupported incoming transactions.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

SMMU implementations supporting other interconnects might define their own set of unsupported incoming transactions.

Note: See section 3.22. A destructive read (read with invalidate), write with directed prefetch, or standalone directed prefetch transaction is not considered to be a discrete Cache Maintenance Operation and is handled differently.

16.7.2.1 SMMUv3.1 control of Cache Maintenance Operations

In SMMUv3.1, STE.DRE controls whether an Invalidate operation is transformed as follows:

Input transaction class	DRE == 0	DRE == 1
Invalidate	Transformed into CleanInvalidate. The operation is treated identically to a CleanInvalidate for permission evaluation.	Eligible for output as Invalidate (if permissions checks allow)

The STE.DRE field applies in this manner when one or more stages of translation are applied. This does not include the case where the only stage of translation is skipped due to STE.S1DSS.

16.7.2.2 SMMUv3.1 permissions model for Cache Maintenance Operations

In SMMUv3,1, the SMMU_S_IDR0.SIF permission check applies to Cache Maintenance Operations. This applies when either translation or bypass occurs.

In SMMUv3.1, when one or more stages of translation are applied, the following permissions are required for Cache Maintenance Operations:

Maintenance operation type	Required permissions	Behavior if permissions not met
Clean, CleanInvalidate, CleanToPersistence	Identical to ordinary read: Requires Read or Execute permission, (depending on input InD and INSTCFG) at privilege appropriate to PnU input and STE.PRIVCFG.	Identical to ordinary read.
Invalidate	To progress as Invalidate, requires both Read-or-Execute permission (depending on input InD and INSTCFG) and Write permission at privilege appropriate to the PnU input and STE.PRIVCFG.	If no Read/Exec permission is available, behavior is identical to an ordinary read. If Read/Exec permission is available but Write permission is not, an Invalidate is transformed into a CleanInvalidate operation.

If a Clean, CleanInvalidate, Invalidate or CleanToPersistence operation leads to a fault, it is recorded as a read (data or instruction, as appropriate to the input InD/INSTCFG). On fault, these operations stall in the same way as an ordinary read transaction if the SMMU is configured for stalling fault behavior. Retry and termination

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

behave the same as for an ordinary read or write transaction. If these transactions are stalled and retried, they are retried as the same transaction type.

Note: The input interconnect might supply all Cache Maintenance Operations as Data.

See section 3.13.7 for information on the behavior of HTTU for Invalidate operations.

When HTTU is enabled for Access flag updates and the translation descriptor and AFFD configuration require it, a Clean, CleanInvalidate, CleanToPersistence or Invalidate operation updates the Access flag. See section 3.13.2.

16.7.2.3 SMMUv3.1 memory types and Shareability for Cache Maintenance Operations

Cache Maintenance Operations do not have a memory type.

In SMMUv3.1, the Shareability of a CMO is determined in the same way to that of an ordinary transaction.

This rule applies to all such operations in all translation and bypass configurations, including:

- Global bypass (attribute set from GBPA).
- STE bypass (whether STE.Config == `0b100` or STE.S1DSS and STE.Config == `0b101` causes skip of the only stage of translation).
- Translation.
- Input attribute is used unmodified.

16.7.3 Treatment of AMBA Exclusives from client devices

The ACE-Lite specification does not permit Exclusive accesses to Inner-shareable or Outer-shareable AMBA domains. Therefore, if an SMMU is an ACE-Lite master, a Non-shareable, or System-shareable, Exclusive access transaction that is translated into an Inner-shareable or Outer-shareable transaction cannot be marked as Exclusive. ARM recommends that such transactions are transformed into non-Exclusives.

For more information about the relationship between AMBA and ARMv8 output attributes, see section 16.7.5.2.

The outcome of such a transformed Exclusive transaction is equivalent to that of an ordinary transaction and depends on whether the transaction experiences a fault and, if it faults, fault configuration. The transaction will experience one of the following:

- Translates without fault, returning the non-exclusive transaction's response to the upstream client device. A response of EXOK is not possible (as the transaction is now non-exclusive). A response of OK will be treated as an exclusive fail by the upstream client device.
- Faults on translation and is terminated with abort. These aborts are reported to the upstream client device in the same way for transformed Exclusive transactions as for regular transactions (for example as a Slave Error).
- Fault on stage 1 translation and be terminated with RAZ/WI semantics because CD.A == 0. This returns a response of OK.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

16.7.4 Treatment of downstream aborts

Some systems might allow a slave device to abort transactions, returning status to the master. Translated transactions initiated by a client device that are aborted in the memory system are not recorded in the SMMU. The abort is returned to the client device, which is responsible for recording and reporting such faults. Aborted transactions that were internally-initiated by the SMMU are recorded by the SMMU if possible to do so.

The event recorded by the SMMU, on one of its accesses being returned with abort status (whether aborted by the interconnect or slave), depends on the type of access:

- STE fetch: F_STE_FETCH
- CD fetch: F_CD_FETCH
- Translation table walk: F_WALK_EABT
- Command queue read entry: GERROR.CMDQ_ERR & Command queue CERROR_ABT
- Event queue access: GERROR.EVENTQ_ABT_ERR
- PRI queue access: GERROR.PRIQ_ABT_ERR
- MSI write: GERROR.MSI_*_ABT_ERR

16.7.5 SMMU and AMBA attribute differences

16.7.5.1 Conversion of AMBA attributes to ARMv8 on input and inferring Inner Attributes

AMBA does not explicitly encode separate inner attributes for an upstream client device. ARM recommends that the inner and outer attributes are considered to be the same as the outer attributes except in the following case for data operations:

- It is IMPLEMENTATION DEFINED whether Non-cacheable with any AxDOMAIN value is treated as iNC-oNC-OSH or whether Non-cacheable with an AxDOMAIN value of NSH/ISH/OSH is treated as an iWB-oNC-{NSH,ISH,OSH} type. In the latter case, it must be considered to be Read-Allocate and Write-Allocate.

Note: Determination of the inner attributes might be used if the downstream interconnect can convey inner attributes.

16.7.5.1.1 Conversion of input attributes from AMBA to ARMv8 architectural attributes

Incoming AMBA attributes are converted to SMMU/ARMv8 architectural attributes as follows:

AMBA attribute	ARMv8 attribute	Notes
Device-Sys non-bufferable	Device-nGnRnE	
Device-Sys bufferable	Device-nGnRE	
Normal-Non-cacheable-Sys (bufferable or non-bufferable)	Normal-iNC-oNC-OSH	
Normal-Non-cacheable-{NSH,ISH,OSH} (bufferable or non-bufferable)	Normal-iNC-oNC-OSH Or Normal-iWB-oNC-{NSH,ISH,OSH}	This is an IMPLEMENTATION DEFINED choice. When the input is treated as iNC-oNC-OSH, RA/WA/TR

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

AMBA attribute	ARMv8 attribute	Notes
		do not exist. Otherwise, RA, WA are 1 and marked non-transient.
Normal-WriteThrough-{NSH,ISH,OSH}	Normal-iNC-oNC-OSH[1] Or Normal-iWT-oWT-{NSH,ISH,OSH}	This is an IMPLEMENTATION DEFINED choice. When the input is treated as iNC-oNC-OSH, RA/WA/TR do not exist. Otherwise, RA,WA are from input and marked non-transient.
Normal-WriteBack-{NSH,ISH,OSH}	Normal-iWB-oWB-{NSH,ISH,OSH}	RA, WA from input. Always marked non-transient.

[1] The conversion between architectural and AMBA attributes might consider WriteThrough to be equivalent to a Normal Non-cacheable type on output and an implementation might, for consistency, apply this strategy on input.

An ACE-Sys input Shareability in considered to be OSH for the purposes of attribute combining and overriding as described in section 13.1.

16.7.5.2 Conversion of ARMv8 attributes to AMBA on output and representation of Shareability

The SMMU specifies the architectural Inner and Outer Cacheability and Shareability attributes. However, in some circumstances there is a non-obvious transformation of these attributes into an AMBA representation:

- The architecture considers any-Device/Normal-iNC-oNC to be OSH, while ACE considers these to be 'Sys'.
- Final attributes of any-Device/Normal-iNC-oNC are presented on AMBA as ACE-Device-Sys/ACE-Normal-Non-cacheable-Sys.
- If the implementation does not transform final attributes of i{WB,WT}-oNC-OSH (inner cacheable of any variety) to Normal-Non-cacheable-SYS as set out in 16.7.5.3 (for example, a different interpretation of attribute mapping is used to that of ARM PE IP) and these attributes are transformed to an ACE cacheable type, the type is represented as ACE-OSH.

16.7.5.2.1 Conversion of ARMv8 architectural attributes to AMBA on output

SMMU/ARMv8 architectural attributes are converted to AMBA attributes on output as follows:

ARMv8 attribute	AMBA attribute	Notes
Device-nGnRnE	Device-Sys non-bufferable	
Device-(n)G(n)RE	Device-Sys bufferable	
Normal-iNC-oNC-OSH	Normal-Non-cacheable-Sys bufferable	Architecturally, a Normal-iNC-oNC-{NSH,ISH} attribute is not possible, only OSH.
Normal-iNC-oWT-{NSH,ISH,OSH}	Normal-Non-cacheable-Sys bufferable	[1]

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

ARMv8 attribute	AMBA attribute	Notes
Normal-iNC-oWB-{NSH,ISH,OSH}	Normal-Non-cacheable-Sys bufferable	(1)
Normal-iWT-oNC-{NSH,ISH,OSH}	Normal-Non-cacheable-Sys bufferable	(1)
Normal-iWT-oWT-{NSH,ISH,OSH}	Normal-Non-cacheable-Sys bufferable	(1)
Normal-iWT-oWB-{NSH,ISH,OSH}	Normal-Non-cacheable-Sys bufferable	(1)
Normal-iWB-oNC-{NSH,ISH,OSH}	Normal-Non-cacheable-Sys bufferable	(1)
Normal-iWB-oWT-{NSH,ISH,OSH}	Normal-Non-cacheable-Sys bufferable	(1)
Normal-iWB-oWB-{NSH,ISH,OSH}	Normal-WriteBack-{NSH,ISH,OSH}	

(1) See 16.7.5.3 below: these transformations correspond to the transformations implemented in the PEs in the system. The outputs shown correspond to ARM Cortex IP. For other PE IP, the interpretations of the ARMv8 attributes are implementation defined.

When a cacheable type is output, AMBA interconnect RA and WA attributes are generated directly from the RA/WA portion of the ARM architectural attribute.

Section 13.1.6 mandates that the SMMU will not output architecturally-inconsistent attributes or attribute combinations that are illegal for the interconnect. For AMBA, the output AxDOMAIN is made consistent with the final AxCACHE value if it is not already. If required, this is made consistent by choosing the highest (most shareable) value of AxDOMAIN that is legal given AxCACHE. Normal Non-cacheable types are always bufferable. The output AxDOMAIN is ACE-Sys if the final attributes are a Device or a Non-cacheable type.

For example, in the case where:

- The SMMU is configured to bypass, SMMU_CR0.SMMUEN == 0.
- SMMU_GBPA.MTCFG == 1, and the input MemAttr is overridden to ‘iWB-oWB’ by SMMU_GBPA.MemAttr.
- SMMU_GBPA.SHCFG == ”use-incoming”.
- An ACE input attribute provides ACE-Device-Sys.

The final output of the SMMU is ACE-WB-OSH.

16.7.5.3 Common interpretation of attribute encoding between SMMU and PE

If interoperation with the ARMv8 PE IP is required, a Normal memory attribute that is not iWB-oWB is transformed to the architectural type iNC-oNC-OSH. See 16.7.5.2, in AMBA systems this is represented as ACE-NC-Sys.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

For example, a final output attribute of iWT-oNC-NSH is converted to iNC-oNC-OSH and is therefore output into an AMBA system as ACE-NC-Sys.

Access attributes of type any-Device are unaffected by this rule.

Otherwise, for interoperation with other PE IP, the transformations between Normal memory attributes that are not iWB-oWB or iNC-oNC and AMBA attributes are IMPLEMENTATION DEFINED.

16.7.6 Far Atomic operations

If an interconnect and SMMU supports client device-initiated Far Atomic operations according to the atomic operations specified in ARMv8.1-A [5], they experience permission checking as though they perform both a read and a write operation. See section 13.1.1 for permission checking and fault reporting. An atomic access is considered to be a write that also performs a read, so is always considered to be Data. The InD attribute and any INSTCFG overrides are ignored for atomic accesses.

Note: For example, a Far Atomic increment to an address in a read-only page must cause a write Permissions Fault (if all other translation requirements are satisfied). If the transaction is configured to stall and is later retried, the entirety of the transaction must be retried atomically. It is prohibited to satisfy the read of data prior to raising a write fault for the update of the data and then use the same read data when the transaction is later retried. The retry must perform the unbroken atomic transaction in one action.

If an upstream interconnect can express this kind of atomic transaction, but the downstream interconnect or system cannot, one of the following occurs:

1. Terminate the transaction in the SMMU with an abort, and record an F_UUT.
2. Support Far Atomic transactions within the SMMU, converting them to local monitor atomic operations using a fully-coherent cache in the SMMU.

In case (1) where far atomics are not supported at all, ARM recommends that the system ensures that upstream devices are not able to emit these transactions (and that software not expect to use them).

16.7.7 AMBA DVM messages with respect to CD.ASET == 1 TLB entries

CD.ASET == 1 affects the interaction of TLB entries with DVM messages in the following ways:

Entries created from StreamWorld == NS-EL1 are not required to be invalidated by:

- Guest OS TLB invalidation by ASID.
- Guest OS TLB invalidation by ASID and VA.

Entries created from StreamWorld == Secure are not required to be invalidated by:

- Secure TLB invalidation by ASID.
- Secure TLB invalidation by ASID and VA.

Entries created from StreamWorld == EL2-E2H are not required to be invalidated by:

- Hypervisor TLB invalidation by ASID.
- Hypervisor TLB invalidation by ASID and VA.

Entries created from StreamWorld == EL2 are not required to be invalidated by:

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Hypervisor TLB invalidation by VA.
- Hypervisor TLB invalidation by ASID.
- Hypervisor TLB invalidation by ASID and VA.

Entries created from StreamWorld == EL3 are not required to be invalidated by:

- EL3 TLB invalidation by VA.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

17 Appendix - The ARM RAS architecture

17.1 Faults, errors, and failures

There are many sources of faults in a system, including both software and hardware faults:

Hardware faults

These faults originate in, or affect, hardware. Hardware faults can be generated from various different sources, and can be further subdivided into:

- Transient faults. These faults include:
  - Interference from a high-energy particle, such as from atomic decay or a cosmic ray.
  - Electrical interference, for example from cross-talk that results from a manufacturing or design defect.
  - Environmental faults, for example overheating or a voltage drop.
- Persistent faults. These faults are non-transient faults, and include:
  - A manufacturing defect that was not found during the manufacturing tests.
  - Faults that are accumulated by a component during its lifetime, for example wear-out, stress, migration, or radiation damage.
  - Faults that occur early in the life of a component during its burn-in period.

Software faults

These faults originate from software bugs, or faults in the input to software.

The RAS architecture describes data corruption faults, which generally occur in memories or on data links. To achieve reliability, availability, and serviceability, the RAS architecture concepts can also be used for handling other types of faults found in systems.

Faults can remain dormant and not affect the operation of a system. When a fault becomes active, it produces an error. Errors propagate within a component and between components. This propagation might halt, for example if the error is overwritten. Where the fault is overwritten, the error is referred to as being masked.

A service failure occurs when an error propagates to the service interface and causes a deviation from correct service. Correct service might encompass:

- Producing correct results.
- Producing results within the time allotted to the task.
- Not divulging secret or secure information.

Therefore:

- A failure is the event of deviation from correct service.
- An error is the deviation.
- A fault is the cause of the error.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Errors that are present but not yet detected are latent or undetected errors. A transaction that carries a latent or undetected error is corrupted.

A component might detect and correct an error:

- If a component fails to detect an error, it is undetected.
- If a component detects and corrects an error, it is a *corrected* error.
- If a producer cannot correct an error, but can continue correct operation, it can *defer* the error.

In the simplest case, an error is propagated if a corrupt value is passed from the producer to the consumer. However, an error is also said to have propagated if:

- A transaction that would not have been permitted to occur had the fault not been activated passes from the producer to the consumer.
- A transaction that would otherwise have occurred does not occur.
- Modified data is lost or any other loss of coherency in a multiprocessor coherent system is observed.
- In some cases, changing the timing and order of transactions is also considered to propagate an error.

If an uncorrected error is received by the consumer as an undetected error, the error has been *silently propagated*. A detected, uncorrected error is uncontained at the component that silently propagates it.

Any error is *uncontainable* if it is uncontained. A detected uncorrected error is *containable* if it is not uncontainable. If the component cannot determine whether a detected error is uncontainable or containable, it must treat it as uncontainable.

Each device sets its own targets for reliability, availability, and serviceability, and uses different techniques to achieve these targets, including:

- Fault prevention.
- Fault removal.
- Fault handling.
- Error handling and recovery.

Fault prevention and fault removal mechanisms are implementation defined. The RAS architecture provides a common programmers' model and mechanisms for fault handling and error recovery.

17.2 Error handling and recovery

For containable uncorrected errors:

- If the error is received by the consumer as a detected error, then the producer has deferred the error to the consumer, creating *a deferred error*.

  A common mechanism for deferring errors is by *poisoning* state. Typically, a first producer detects the error on data that it is about to write to a first consumer that will update the storage location. The first producer

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

signals a Deferred error on the write, invalidating its copy of the data, and the first consumer poisons the location.

If the location is later accessed by a second consumer, the poison is detected and signaled by the first consumer, which acts as the second producer, as an Uncorrected error. Alternatively, the first consumer might defer the error again to the second consumer.

- If a component transports or contains corrupted data, and the corrupted data is marked as corrupted, then the component is propagating the error and no further action is required.

  Note: If a detected error is allocated into a cache without being signaled in the cache as having an error, it might be silently propagated and therefore, in this case, is uncontainable.

- If a consumer of an error does not require the corrupted data to make progress, and therefore can recover with no action, the error is a *restartable error*. It is implementation defined whether such errors are signaled to software.

  Note: An error on a speculative read that has not updated the state of the component is often containable and restartable.

- If the consumer can take action to recover from the error, possibly requiring software intervention, before corrupting its state, but requires the corrupted data to make progress, the error is a *recoverable error*.
- If the error has corrupted the state of the component so that recovery of the component is not possible, the error is an *unrecoverable error* for the affected component.

An error that is uncontainable at the component might still be containable at the system level.

Note: Hardware cannot know the capabilities of software. Therefore, a component might report an error as restartable or recoverable, and software must implement recovery.

A PE consumes an error as an External Abort or SError interrupt exception. Other system components might signal errors to software using an error recovery interrupt that is usually sent to an interrupt controller. See also Error recovery interrupt and Error recovery and fault handling signaling

Note: Reporting an error as containable allows software to contain the error. It does not mean that hardware has contained the error.

17.3 Fault handling

When an error is detected by a component that supports fault reporting, it records the error in an error record and might generate a fault handling interrupt that is usually sent to an interrupt controller, see also. See also Error recovery interrupt and Error recovery and fault handling signaling

Note: A fault handling interrupt is logically separate from the error recovery interrupt described in Error handling and recovery. This is because the processes of fault handling and error recovery are logically separate. Controls are provided to enable and disable the fault handling interrupt.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

17.4 Nodes

In the RAS architecture, the component that detects an error is called a node. The architecture defines the following common features for a node. A node might implement some or all of these features, and, if implemented, might implement controls to enable or disable the features in ERR<n>CTLR.

Error detection or correction, or both

The level of error correction and detection that is implemented at a node is implementation defined. A node might include a control to disable error detection and correction, for example while software initializes the node.

Fault handling interrupt

Asynchronous reporting of all or some detected errors of an interrupt. That is, all Corrected errors, Deferred errors, and Uncorrected errors. It is implementation defined whether a node provides a single control for all errors, or a first control for Corrected errors and a second control for all other detected errors.

See also Fault handling interrupt.

Corrected error counting

It is implementation defined whether a node that provides error correction implements a counter for counting Corrected errors. Software might poll the error counter or initialize the counter with a negative threshold value and receive an interrupt when the counter overflows.

It is implementation defined which Corrected errors are counted.

For example, if an implementation is capable of correcting multiple errors in a single cycle, although this is statistically very unlikely compared to the overall corrected error FIT (failure-in-time) rate, the implementation might decide to only count one of the Corrected errors.

See also Standard format Corrected error counter.

In-band Uncorrected error signaling (external aborts)

In-band signaling of detected Uncorrected errors to the consumer of the error. This is also referred to as an external abort. Corrected errors and Deferred errors are not reported by such means.

Note: In this overview, *external abort* refers to any in-band error signaling on an interface. This includes error signaling on a memory interface from a PE, which might generate External Abort exceptions. However, other interfaces can also include in-band error signaling, and so any use of external abort in this section also includes these interfaces.

See also In-band error signaling.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Uncorrected error recovery and handling interrupt

Asynchronous (out-of-band) reporting of detected Uncorrected errors by an interrupt. The interrupt can be used for error recovery, fault handling, or both. Corrected errors are not reported by this means. It is implementation defined whether the node provides a control to enable Deferred errors to be reported in this way. If the control is not supported, Deferred errors are not reported by this means.

The error recovery interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

Note:

- An interrupt is always asynchronous and is routed by the interrupt controller. If implemented as a Shared Peripheral Interrupt (SPI), it can be routed to any PE as an IRQ or FIQ interrupt. For more information about SPIs, see the *ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0*.
- If a detected error is signaled by an error recovery interrupt and not as an external abort, then the consumer does not consume the detected error.
- For brevity, this is called the error recovery interrupt in other parts of this appendix. See also Error recovery interrupt.

Records

A node implements one or more records. When an error is detected, syndrome about the error is written to a record.

See Standard error record.

Throughout this architecture, where it is required that a node returns an external abort, defers an error, or generates an interrupt, it is assumed that the feature is both implemented and enabled.

17.4.1 Synchronization and error record accesses

When a node detects an error it updates the error record registers and might generate one or more of:

- A fault handling interrupt.
- An error recovery interrupt.
- An in-band error response.

Each of these might generate an exception at a PE. After taking an exception generated by such a signal from a node, if the PE reads the error record registers at the node, the read must return the updated values.

For memory-mapped registers, this assumes the memory-mapped registers are mapped as a Device type that does not permit read speculation.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

17.5 Standard error record

An error record holds syndrome for the error:

- Status of the error.
- An address, if applicable.
- Optionally, counters for software to poll the rate of Corrected errors.
- Information to identify a field replaceable unit (FRU) and locate the error within the FRU.
- Other IMPLEMENTATION DEFINED information.

The error record registers might also contain control registers for error detection, correction, and reporting at the node.

The RAS extension defines a standard error record and a mechanism to access error records as System registers. The standard error record contains:

- Controls for common features, and an identification mechanism for these controls. See Nodes for more information about common features.

  For each node it IS IMPLEMENTATION DEFINED whether the fault and error reporting mechanisms apply to both reads and writes, or can be individually controlled for reads and writes.
- A status register for common status fields, such as the type and coarse characterization of the error.
- An address register, if applicable.
- Standard mechanisms for Corrected error counters for software to poll the rate of Corrected errors.
- IMPLEMENTATION DEFINED controls and identification registers.
- IMPLEMENTATION DEFINED status registers. ARM recommends that these are used to identify a FRU and locate the error within the FRU.

Error records can be accessed using memory-mapped interfaces. Memory-mapped view of an error record defines a reusable format for memory-mapped registers. Use of the reusable format is optional.

Error records must be preserved over error recovery reset, allowing diagnosis after system failure.

ARM recommends that all error records are remotely accessible for access by all PEs in a system, or by a Baseboard Management Controller (BMC) or System Control Processor (SCP). The remote access mechanism is IMPLEMENTATION DEFINED, but might use CoreSight-like interfaces. ARM recommends that remote access is possible when the rest of the system is in a fail state, for example when the system has locked up.

17.5.1 Security and virtualization

If a component with memory-mapped fault handling registers processes Secure data, then the fault handling registers must either:

- Be visible only to Secure accesses.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Provide a reduced functionality to Non-secure state that does not affect operation in Secure state, or does not provide visibility of Secure data, or both.

If a component with memory-mapped fault handling registers processes only Non-secure state, then it is IMPLEMENTATION DEFINED whether:

- The fault handling registers are visible to both Non-secure and Secure accesses.
- It is configurable whether the fault handling registers are visible to Non-secure accesses.
- The fault handling registers are visible only to Secure accesses.

ARM recommends that it is configurable whether memory-mapped fault handling registers are visible to Non-secure accesses.

Note: The mechanism by which Secure firmware prevents Non-secure accesses to memory-mapped fault handling registers is IMPLEMENTATION DEFINED.

If a component provides multiple virtual peripherals for use by multiple guest operating systems under a hypervisor, ARM recommends that each virtual peripheral has a set of fault handling registers to control error recovery and fault handling for that virtual machine. This allows a hypervisor to isolate errors to a given virtual machine. An additional set of fault handling registers might be provided for use by the hypervisor or Secure monitor, and for errors that cannot be isolated.

17.5.2 Multiple records per node

Each node contains at least one error record. Where a node implements multiple error records, it is IMPLEMENTATION DEFINED whether this is used by the node:

- To record different types of error in different records. For example, a node detects errors in multiple memories, such as DIMMs, and implements one record for each memory.

  Note: In this case the multiple records behave as a single node because they share control and identification registers.
- To record multiple errors. For example, on detecting an error the node chooses a free record from the multiple records.

17.5.3 Error types in the error record

The highest priority recorded error type is recorded in the ERR<n>STATUS.{CE, DE, UE, UET} fields, as shown in Table 1.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

ERR<n>STATUS fields					Highest priority error type	Mnemonic
V	CE	DE	UE	UET		
0	X	X	X	`0bXX`	None	-
1	`0b00`	0	0	`0bXX`		
1	!= `0b00`	0	0	`0b00`	Corrected error	CE
1	X	1	0	`0b00`	Deferred error	DE
1	X	X	1	`0b10`	Restartable uncorrected error	UEO
1	X	X	1	`0b11`	Recoverable uncorrected error	UER
1	X	X	1	`0b01`	Unrecoverable uncorrected error	UEU
1	X	X	1	`0b00`	Uncontainable uncorrected error	UC

Table 1: Encoding the highest priority error

Note: Non-zero X bits record the highest priority error, and additionally might record previously recorded errors.

For example, if ERR<n>STATUS.{CE, DE, UE, UET} ==`{0b01, 1, 0, 0b00}`, then the highest priority error is deferred, but at least one corrected error might have been previously recorded.

The error types that are implemented at a node are IMPLEMENTATION DEFINED. An implementation might only include a simplified subset of these error types. A node can always record:

- UEO as any of UER, UEU, or UC.
- UER as either UEU or UC.
- UEU as UC.

17.5.4 Writing the error record

When a new error is detected, the node:

- Sets or modifies ERR<n>STATUS.{CE, DE, UE, UET} to indicate the type of the new detected error, see Error types in the error record.

  Note: ERR<n>STATUS.{DE, UE} are single-bit fields that are set to 1. ERR<n>STATUS.CE is a 2-bit field that might be modified.

- Either:
  - Overwrites the error record with the syndrome for the new error, if it has a higher priority than the previous highest priority recorded error.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

- Keeps the syndrome for the previous error, if the new error has a lower priority than the previous highest priority recorded error.

  Note: One of ERR<n>STATUS.{CE, DE, UE} is always set or modified even if the new error syndrome is discarded.

- Counts the error, if it is a Corrected error and a counter is implemented.

  Note: A counter for Corrected errors is optional, see ERR<n>MISC0.

17.5.5 Prioritizing errors

Overwriting depends on the type of the previous highest priority error and the type of newly recorded error, as shown in Table 2. In this table, the row and column headings use the mnemonics for Table 1 and the following other abbreviations are used:

K Keep. Keep the previous error syndrome. It is implementation defined whether ERR<n>STATUS.OF is set to 1 or unchanged.

O Overflow. Keep the previous error syndrome and set ERR<n>STATUS.OF to 1. This means that software can detect that a secondary error of the same type was discarded.

W Overwrite. Record the new error syndrome. It is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to 0 or unchanged.

CK Count and keep. Count CE if a counter is implemented, and keep the previous error syndrome. If the counter overflows, or no counter is implemented, it is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to 1 or unchanged.

CWK Count and overwrite or keep. The behavior is IMPLEMENTATION DEFINED and described by the value of ERR<n>FR.CEO:

- **0** Count CE if a counter is implemented, and keep the previous error syndrome. If the counter overflows, or if no counter is implemented, ERR<n>STATUS.OF is set to 1.
- **1** Count CE. If ERR<n>STATUS.OF == 1 before the CE is counted, keep the previous syndrome. Otherwise record the new error syndrome. If the counter overflows, ERR<n>STATUS.OF is set to 1.

CW Count and overwrite. Count CE if a counter is implemented, and overwrite. If a counter is implemented and overflows, ERR<n>STATUS.OF is set to an unknown value. Otherwise, it is IMPLEMENTATION DEFINED whether ERR<n>STATUS.OF is set to 0 or unchanged.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

WO Overwrite and overflow. ERR<n>STATUS.OF is set to 1.

Previous error type	New detected error type					
	CE	DE	UEO	UER	UEU	UC
-	CW	W	W	W	W	W
CE	CWK	W	W	W	W	W
DE	CK	O	W	W	W	W
UEO	CK	K	O	WO	WO	WO
UER	CK	K	O	O	WO	WO
UEU	CK	K	O	O	O	WO
UC	CK	K	O	O	O	O

Table 2: Rules for overwriting error records

17.5.6 Overwriting the error syndrome

When an old record is overwritten:

- Bits shown as X in Table 1for the new error are unchanged. This allows software to detect that the syndrome for a previous error was overwritten.
- The ERR<n>STATUS.{ER, PN, IERR, SERR} syndrome fields are written with the syndrome for the new error.
- If there is an address syndrome for the new error, ERR<n>STATUS.AV is set to 1 and the address that is written to ERR<n>ADDR. Otherwise ERR<n>STATUS.AV is set to 0.
- If there is other miscellaneous syndrome for the new error, it is written to the ERR<n>MISC0 or ERR<n>MISC1 registers and ERR<n>STATUS.MV is set to 1.
- If there is no additional miscellaneous syndrome for the new error that is written to the ERR<n>MISC0 or ERR<n>MISC1 registers, then it is IMPLEMENTATION DEFINED whether ERR<n>STATUS.MV is set to 0 or unchanged:
  - If software can determine from the ERR<n>MISC0 and ERR<n>MISC1 contents that the syndrome is not related to the highest priority error, the MV bit is unchanged.
  - Otherwise the MV bit is cleared to 0.
- ERR<n>STATUS.V, the Valid bit, is set to 1.

When software has processed an error record it must clear the Valid bit for that record so that it can record new errors. Software must only clear those fields that it believes to be set in the record. Therefore, hardware can

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

correctly detect if a further error has occurred since software read the record, and prevent the Valid bit from being cleared.

17.5.7 Keeping the previous error syndrome

When an old record is kept:

- ERR<n>STATUS.{PN, IERR, SERR} are unchanged. ERR<n>ADDR and ERR<n>STATUS.AV are unchanged. If the new error is an Uncorrected error, then ERR<n>STATUS.UET is unchanged.
- It is IMPLEMENTATION DEFINED whether ERR<n>MISC0 or ERR<n>MISC1, or both, are updated. The content of the ERR<n>MISC0 and ERR<n>MISC1 registers is IMPLEMENTATION DEFINED. As such, it is possible that some of the information about an otherwise discarded error is recorded in these registers. If data is written to ERR<n>MISC0 or ERR<n>MISC1 then ERR<n>STATUS.MV is set to 1. Software must be able to determine which data relates to which type of error.

The applicable one of ERR<n>STATUS.{CE, DE, UE} is set or modified even if the new error syndrome is discarded. DE and UE are single-bit fields that are set to 1. CE is a 2-bit field that might be modified, as in the following example.

Example: On detecting a first Corrected error, CE is set to `0b10`. On detecting a second Corrected error, the new error is discarded, but CE is updated to `0b11` (persistent) if this error is detected at the same location to the first error, or `0b01` (transient) otherwise. Support for persistent error detection is IMPLEMENTATION DEFINED.

17.5.8 Detecting multiple errors

If a node detects multiple errors simultaneously, it is IMPLEMENTATION DEFINED whether the nodes behave:

- As if all errors were recorded, in any order. In this case, the prioritization rules mean that the highest priority error will be recorded in the syndrome registers. However, the final value of the syndrome register might depend on the logical order in which the errors were recorded.
- As if the highest priority error was recorded and one or more of the lower priority errors were not recorded.

If multiple errors are corrected simultaneously, and a Corrected error counter is implemented, it is IMPLEMENTATION DEFINED whether all the Corrected errors are counted.

ARM recommends that if multiple simultaneous errors are statistically likely when, (compared to the overall FIT rate), then the implementation behaves as if all errors are recorded.

17.5.9 Standard format Corrected error counter

The architecture defines standard formats for a Corrected error counter (CE counter) that can be indicated in the ERR<n>FR register when implemented in ERR<n>MISC0:

- 8 bits, comprising a 7-bit CE counter plus a single sticky overflow bit.
- 16 bits, comprising a 15-bit CE counter plus a single sticky overflow bit.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and the overflow bit is set.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

An implementation might include a pair of counters. In such an arrangement the first counter, called the repeat counter, counts errors at the same location. Errors in other locations are counted in a second error counter, called the other counter. The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set.

Note: IMPLEMENTATION DEFINED forms of counter, including other sizes, other overflow models, and other ERR<n>MISC0 or ERR<n>MISC1 locations might be implemented.

17.6 Error recovery interrupt

If an error recovery interrupt is implemented by a node, then the set of controls for enabling error recovery interrupts is IMPLEMENTATION DEFINED:

- A control for enabling the error recovery interrupt on Deferred errors, ERR<n>CTLR.DUI might be implemented:
  - If the DUI control is implemented, it enables the error recovery interrupt for Deferred errors.
  - If the DUI control is not implemented, the error recovery interrupt is never generated for Deferred errors.
- A control for enabling the error recovery interrupt on Uncorrected errors, ERR<n>CTLR.UI might be implemented:
  - If the UI control is implemented, it enables the error recovery interrupt for Uncorrected errors.
  - If the UI control is not implemented, the error recovery interrupt is always enabled for Uncorrected errors.

If an error recovery interrupt is not implemented by a node, these controls are not implemented.

For each implemented control, it is further implementation defined whether there is a single control or whether there are separate controls for reads and writes, for example, ERR<n>CTLR.{RUI, WUI} in place of ERR<n>CTLR.UI.

Software uses ERR<n>FR.{DUI, UI} to discover which controls are implemented.

See also Error recovery and fault handling signaling

17.7 Fault handling interrupt

If a fault handling interrupt is implemented by a node, then the set of controls for enabling fault handling interrupts is IMPLEMENTATION DEFINED:

- A control for generating the fault handling interrupt on Corrected errors, ERR<n>CTLR.CFI, might be implemented. If the control is implemented:
  - The CFI control enables the fault handling interrupt for *corrected error events*.
  - The ERR<n>CTLR.FI control must also be implemented and enables the fault handling interrupt for Deferred and Uncorrected errors.
- If the CFI control is not implemented, a control for generating the fault handling interrupt on all detected errors, ERR<n>CTLR.FI, might be implemented:
  - If the FI control is implemented, it enables the fault handling interrupt for all corrected error events, Deferred and Uncorrected errors.
  - If the FI control is not implemented, the fault handling interrupt is always enabled for all corrected error events, Deferred and Uncorrected errors.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If a fault handling interrupt is not implemented by a node, these controls are not implemented.

A *corrected error event* is either:

- When the counter overflows and sets the overflow bit, if the node implements a Corrected error counter, see Standard format Corrected error counter.
- Each detected Corrected error, otherwise.

For each implemented control, it is further IMPLEMENTATION DEFINED whether there is a single control or whether there are separate controls for reads and writes, for example ERR<n>CTLR.{RFI, WFI} instead of ERR<n>CTLR.FI.

Software uses ERR<n>FR.{CFI, FI} to discover which controls are implemented.

The fault handling interrupt is generated when the node detects an error, even if the error syndrome is discarded because the error record already records a higher priority error, see Writing the error record and Error recovery and fault handling signaling.

17.8 In-band error signaling

If support for in-band error signaling, also referred to as external aborts, is implemented by a node, then controls for signaling external aborts, ERR<n>CTLR.UE might be implemented:

- If the control is implemented, it enables external abort signaling for all Uncorrected errors.
- If the control is not implemented, then external aborts are always enabled for all Uncorrected errors.

If external aborts are not implemented by a node, these controls are not implemented.

For this control, it is further IMPLEMENTATION DEFINED whether there is a single control or separate ERR<n>CTLR.{RUE, WUE} controls for reads and writes.

Software uses ERR<n>FR.UE to discover which controls are implemented.

When the node signals an Uncorrected error using an external abort, it sets ERR<n>STATUS.ER to 1.

17.9 Error recovery and fault handling signaling

Error recovery and fault handling interrupts are normally routed using the interrupt controller (typically a GIC).

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The GIC generates either an IRQ or FIQ interrupt exception at the PE depending on the group assigned to the interrupt. To allow it to be delivered to firmware as a high priority group 0 interrupt, error recovery and fault handling interrupts must be signaled as either *Shared Peripheral Interrupts* (SPIs) or *Private Peripheral Interrupts* (PPIs).

Note:

- A PPI can only be taken by the PE that it is private to. Any error that is not private to a PE or might need to be processed by another PE (such as an SCP), must generate an SPI.
- LPIs can only be taken as group 1 interrupts.
- An implementation might provide low-latency interrupts for error recovery. The latency of error recovery interrupts is outside the scope of the architecture.
- An in-band error (external abort) might generate an SError interrupt (SEI) exception if it is consumed at a PE. In this case, the external abort is routed to the PE in-band with the data, and the PE generates the SEI exception. The GIC is not involved in routing external aborts

It is IMPLEMENTATION DEFINED whether each node uses independent interrupts for fault handling and error recovery, or shares interrupts with fault handling and/or error recovery interrupts of other nodes. If multiple nodes share an interrupt, fault handling software must use the fault record registers to determine the source of the fault.

Example: A single component implements multiple error sources, such as multiple independent caches. Each error source is a node and has a set of fault record registers, but all share the same fault handling interrupt. This fault handling interrupt might also be shared with error recovery interrupts.

It is IMPLEMENTATION DEFINED whether interrupts are edge-triggered or level-sensitive.

If the interrupt is level-sensitive, it is asserted by the node while enabled in ERR<n>CTLR and the appropriate error flag or flags in the corresponding ERR<n>STATUS register or (if implemented and applicable) overflow bit, or both, in the Corrected error counter. That is:

- A fault handling interrupt remains asserted while any of:
  - Fault handling interrupts on all detected errors are enabled, the ERR<n>STATUS.V bit is set to 1, and either or both of the ERR<n>STATUS.{DE,UE} bits are set to 1.
  - Fault handling interrupts on corrected errors are enabled and either:
    - If the node implements a Corrected error counter, both the ERR<n>STATUS.V bit and the counter overflow bit are set to 1.
    - Otherwise, both the ERR<n>STATUS.V bit is set to 1 and the ERR<n>STATUS.CE field is nonzero.
- An error recovery interrupt remains asserted while any of the following are true:
  - Error recovery interrupts on Uncorrected errors are enabled and both the ERR<n>STATUS.V and ERR<n>STATUS.UE bits are set to 1.
  - Error recovery interrupts on Deferred errors are enabled and both the ERR<n>STATUS.V and ERR<n>STATUS.DE bits are set to 1.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If the interrupt is edge-triggered, then an enabled interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error. See Writing the error record,

Error recovery interrupt, and Fault handling interrupt describe the controls for enabling these interrupts.

When an error is detected and recorded, or an interrupt becomes enabled, the state of the interrupts must be updated in finite time.

The system architecture, for example the Server Base System Architecture, must define, or define mechanisms, to discover:

- Which SPIs and PPIs are used for error recovery and fault handling interrupts.
- Which SGIs are used to delegate errors and faults from firmware to other agents.

For more information about the Server Base System Architecture, see the *ARM® Server Base System Architecture*.

17.10 Error recovery reset

A system comprises multiple power and logical domains, each of which might implement one or more reset signals.

Two classes of reset are defined:

- Cold reset is asserted to a component when it transitions from a powerdown state to a power up state. Cold reset initializes the component to a known initial state. No state is preserved from the previous powerdown state.
- Error recovery reset is an optional reset that might be applied at any other time. System Error Recovery reset initializes the component to a known state. Unlike Cold reset, any recorded error syndrome information is preserved over a System Error Recovery reset.

The way these resets map to other resets is IMPLEMENTATION DEFINED. The mechanisms for asserting resets are IMPLEMENTATION DEFINED.

For a PE, the Error Recovery reset might be implemented by the Warm reset defined by the ARMv8 architecture. If Warm reset is implemented, it must preserve the error records in the PE.

ARM strongly recommends that all components implement an Error Recovery reset.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

17.11 Memory-mapped view of an error record

This section defines the optional memory-mapped registers that might be implemented by a node in a system implementing the RAS extensions. There are two views:

- Single record (within a component) describes a reusable format for a component that implements a single error record. This might be implemented as part of the control registers for a component.
- Group of records describes a reusable format for a component that implements several records in a group relating to one or more nodes.

Register index describes a group with up to 56 records, the highest number that can be contained in a 4KB group.

17.11.1 Supported access sizes

The memory-mapped view of a record or group of records is a mix of:

- 32-bit registers at word-aligned locations.
- 64-bit registers at doubleword-aligned locations.

The memory access sizes supported by any peripheral is IMPLEMENTATION DEFINED by the peripheral. For accesses to the memory-mapped view of a record or group or records, implementations must support:

- Word-aligned 32-bit accesses to access 32-bit registers or either half of a 64-bit register.
- Doubleword-aligned 64-bit accesses to access 64-bit registers.

These accesses must be single-copy atomic at the peripheral.

The following accesses are not supported:

- Byte.
- Halfword.
- Unaligned word (such accesses are not word single-copy atomic).
- Unaligned doubleword (such accesses are not doubleword single-copy atomic).
- Doubleword accesses to a pair of 32-bit locations that are not a doubleword-aligned pair forming a 64-bit register.
- Quad-word or higher.
- Exclusives.

For each of these access types it is UNPREDICTABLE whether:

- The access generates a fault handling interrupt or not.
- A read is ignored or not, in terms of its side-effects. A read returns UNKNOWN data.
- A write is ignored or sets the accessed register or registers to UNKNOWN .

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

For memory-mapped accesses from an ARM architecture PE, the size of an access made by an instruction is determined by the single-copy atomicity rules for the instruction, the type of instruction, and the type of memory accessed.

ARMv8 does not require the size of each element that is accessed by a multi-register load or store instruction to be identifiable by the memory system beyond the PE. Any memory-mapped access to a peripheral register is defined to beyond the PE.

Example: Two Doubleword instructions made to consecutive doubleword-aligned locations will generate a pair of single-copy atomic doubleword reads, but if made to Normal memory or Device-GRE memory might appear as a single quadword access which is not supported by the peripheral.

Software must use a Device-nGRE or tighter memory-type and use only single register load and store instructions to create memory accesses that are supported by the peripheral, For more information, see the *ARM® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile*.

17.12 Reset values

Unless otherwise stated, the reset values of all registers are IMPLEMENTATION DEFINED and might be UNKNOWN. Unless explicitly stated, the reset value refers to both Error Recovery and Cold reset.

Where a Cold reset value is explicitly stated, the register is unchanged on an Error Recovery reset.

17.12.1 Writes to ERR<n>STATUS

```
// ERRSTATUS[] (assignment form)
// ============================
// For a system register, n = UInt(ERRSELR_EL1.SEL)

ERRSTATUS[integer n] = bits(32) w

    STATUS = _ERRSTATUS[n];                                   // previous value (physical register)

    c = (STATUS<31:20> AND NOT(w<31:20>)):Zeros(4):w<15:0>;   // candidate value

    if c.OF == '1' then c.<UE,DE,CE> = STATUS.<UE,DE,CE>;     // do not clear UE/DE/CE if OF set

    if !IsZero(c.<UE,DE,CE>) then c.V = STATUS.V;             // do not clear V if UE/DE/CE set

    if (c.UE != '0' ||                                        // do not clear syndrome if not
        (STATUS.UE == '0' && c.DE != '0') ||                  // clearing highest priority error
        (STATUS.<UE,DE> == '00' && c.CE != '00')) then

        c.<AV,ER,MV,PN,UET,IERR,SERR> = STATUS.<AV,ER,MV,PN,UET,IERR,SERR>;

    _ERRSTATUS[n] = c;
```


Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

17.13 Register index

Offset	Access	Size	Register	Description
0x000+64×n	RO	64	ERR<n>FR	Error Record Feature Register
0x008+64×n	RW	64	ERR<n>CTLR	Error Record Control Register
0x010+64×n	RW	64	ERR<n>STATUS	Error Record Primary Status Register
0x018+64×n	RW	64	ERR<n>ADDR	Error Record Address Register
0x020+64×n	RW	64	ERR<n>MISC0	Error Record Miscellaneous Register 0
0x028+64×n	RW	64	ERR<n>MISC1	Error Record Miscellaneous Register 1
0xE00+8×n	RO	64	ERRGSR<n>	Error Group Status Register
0xE80+8×n	RW	64	ERRIRQCR<n>	Error Interrupt Configuration Register
0xFBC	RO	32	ERRDEVARCH	Device Architecture Register
0xFC8	RO	32	ERRIDR	Error Record ID Register

Table 3: Memory-mapped register map

Offset	Name	Field	Value	Meaning
0xFF0	RAS_CIDR0, Component ID 0	[7:0]	0x0D	Preamble
0xFF4	RAS_CIDR1, Component ID 1	[7:4]	0xF	Class
		[3:0]	0x0	Preamble
0xFF8	RAS_CIDR2, Component ID 2	[7:0]	0x05	Preamble
0xFFC	RAS_CIDR3, Component ID 3	[7:0]	0xB1	Preamble

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Offset	Name	Field	Value	Meaning
0xFE0	RAS_PIDR0, Peripheral ID0	[7:0]	IMPLEMENTATION DEFINED	PART_0, bits[7:0] of the Part number
0xFE4	RAS_PIDR1, Peripheral ID1	[7:4]	IMPLEMENTATION DEFINED	DES_0, bits[3:0] of the JEP106 Designer code
		[3:0]	IMPLEMENTATION DEFINED	PART_1, bits[11:8] of the Part number
0xFE8	RAS_PIDR2, Peripheral ID2	[7:4]	IMPLEMENTATION DEFINED	REVISION
		[3]	1	JEDEC-assigned value for DES always used
		[2:0]	IMPLEMENTATION DEFINED	DES_1, bits[6:4] of the JEP106 Designer code
0xFEC	RAS_PIDR3, Peripheral ID3	[7:4]	IMPLEMENTATION DEFINED	REVAND
		[3:0]	IMPLEMENTATION DEFINED	CMOD
0xFD0	RAS_PIDR4, Peripheral ID4	[7:4]	IMPLEMENTATION DEFINED	SIZE
		[3:0]	IMPLEMENTATION DEFINED	DES_2, JEP106 Designer continuation code
0xFD4	RAS_PIDR5, Peripheral ID5	-	RES0	Reserved
0xFD8	RAS_PIDR6, Peripheral ID6	-	RES0	Reserved
0xFDC	RAS_PIDR7, Peripheral ID7	-	RES0	Reserved

Table 4: CoreSight ID register map

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

18 Appendix - The ARM RAS registers

18.1.1 ERR<*n*>ADDR, Error Record Address Register

The ERR<*n*>ADDR characteristics are:

Purpose

If an error has an associated address, this must be written to the address register when the error is recorded. It is IMPLEMENTATION DEFINED how the recorded addresses map to the software-visible physical addresses. Software might have to reconstruct the actual physical addresses using the identity of the node and knowledge of the system.

Usage constraints

Ignores writes if ERR<n>STATUS.AV is set to 1.

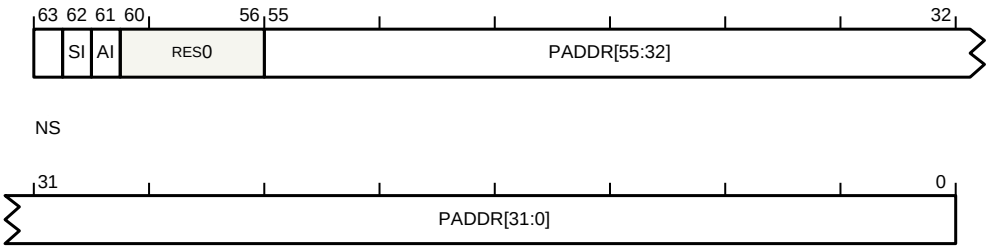
Configurations

Always implemented.

Attributes

64-bit read/write register.

The ERR<*n*>ADDR bit assignments are:



NS, bit [63]

Non-secure attribute. The possible values of this bit are:

`0` The address is Secure.

`1` The address is Non-secure.

SI, bit [62]

Secure Incorrect. Indicates whether the NS bit is valid. The possible values of this bit are:

`0` The NS bit is correct. That is, it matches the programmers' view of the Non-secure attribute for this recorded location.

`1` The NS bit might not be correct, and might not match the programmers' view of the Non-secure attribute for the recorded location.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

It is IMPLEMENTATION DEFINED whether this bit is read-only or read/write.

AI, bit [61]

Address Incomplete or incorrect. Indicates whether the PADDR field is a valid physical address. The possible values of this bit are:

`0` The PADDR field is a valid physical address. That is, it matches the programmers' view of the physical address for this recorded location.

`1` The PADDR field might not be a valid physical address, and might not match the programmers' view of the physical address for the recorded location.

It is IMPLEMENTATION DEFINED whether this bit is read-only or read/write.

Bits [60:56]

Reserved. This field is RES0.

PADDR, bits [55:0]

Address. Unimplemented bits are RES0.

18.1.2 ERR<*n*>CTLR, Error Record Control Register

The ERR<*n*>CTLR characteristics are:

Purpose

The error control register contains enable bits for the node that writes to this record:

- Enabling error detection and correction.
- Enabling an error recovery interrupt.
- Enabling a fault handling interrupt.
- Enabling error recovery reporting as a read or write error response.

For each bit, if the selected node does not support the feature, the bit is RES0. The definition of each record is IMPLEMENTATION DEFINED.

Usage constraints

None.

Configurations

For some controls, it is IMPLEMENTATION DEFINED whether ERR<*n*>CTLR contains a single combined read/write control or separate read and write controls. This register description shows two possible combinations:

- All controls are combined.
- All controls are separate.

However, this is IMPLEMENTATION DEFINED for each control.

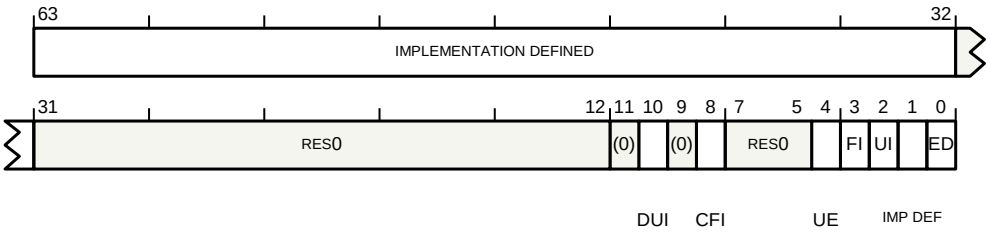
If a single node has multiple records, then only the first record has a populated error control register. For other records of the same node, the error control register is RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

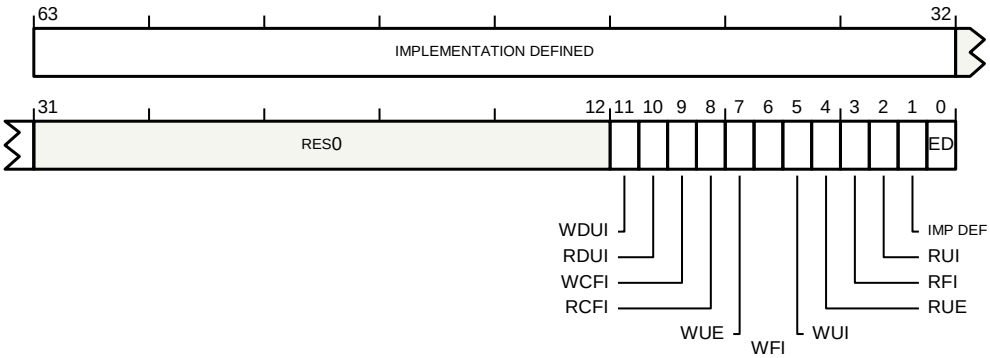
Attributes

64-bit read/write register located at offset `0x008` + `64`×*n*.

When combined read/write control, the ERR<*n*>CTLR bit assignments are:



When separate read/write controls, the ERR<*n*>CTLR bit assignments are:



Bits [63:32]

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Bits [31:12]

Reserved. This field is RES0.

WDUI, bit [11], when separate read/write controls

Error recovery interrupt for deferred errors on writes. As the DUI bit, except for writes only.

Bits [11,9,7:5], when combined read/write control

Reserved. This field is RES0.

DUI, bit [10], when combined read/write control

Error recovery interrupt for deferred errors enable. When enabled the error recovery interrupt is generated for all detected Deferred errors. The possible values of this bit are:

`0` Error recovery interrupt not generated for deferred errors.

`1` Error recovery interrupt generated for deferred errors.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error. If the node does not support this control, this bit is RES0.

Note: Applies to both reads and writes.

RDUI, bit [10], when separate read/write controls

Error recovery interrupt for deferred errors on reads. As the DUI bit, except for reads only.

WCFI, bit [9], when separate read/write controls

Fault handling interrupt for corrected errors on writes. As for CFI, except for writes only.

CFI, bit [8], when combined read/write control

Fault handling interrupt for corrected errors enable. When enabled:

- If the node implements a Corrected error counter, the fault handling interrupt is only generated when the counter overflows and the overflow bit is set. See ERR<n>MISC0.
- Otherwise the fault handling interrupt is also generated for all detected Corrected errors.

The possible values of this bit are:

`0` Fault handling interrupt not generated for corrected errors.

`1` Fault handling interrupt generated for corrected errors.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error. If the node does not support this control, this bit is RES0.

Note: Applies to both reads and writes.

RCFI, bit [8], when separate read/write controls

Fault handling interrupt for corrected errors on reads. As for CFI, except for reads only.

WUE, bit [7], when separate read/write controls

Error reporting on writes. As the UE bit, except for writes only.

WFI, bit [6], when separate read/write controls

Fault handling interrupt on writes. As the FI bit, except for writes only.

WUI, bit [5], when separate read/write controls

Error recovery interrupt on writes. As the UI bit, except for writes only.

UE, bit [4], when combined read/write control

In-band Uncorrected error reporting enable. When enabled, responses to transactions that detect an uncorrected error that cannot be deferred are signaled as a detected error (external abort). The possible values of this bit are:

`0` External abort response for uncorrected errors disabled.

`1` External abort response for uncorrected errors enabled.

If the node does not support this control, this bit is RES0.

Note: Applies to both reads and writes.

RUE, bit [4], when separate read/write controls

Error reporting on reads. As the UE bit, except for reads only.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

FI, bit [3], when combined read/write control

Fault handling interrupt enable. When enabled:

- The fault handling interrupt is generated for all detected Deferred errors and Uncorrected errors.
- If the fault handling interrupt for corrected errors control is not implemented:
  - If the node implements a Corrected error counter, the fault handling interrupt is also generated when the counter overflows and the overflow bit is set.
  - Otherwise the fault handling interrupt is also generated for all detected Corrected errors.

The possible values of this bit are:

`0` Fault handling interrupt disabled.

`1` Fault handling interrupt enabled.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error. If the node does not support this control, this bit is RES0.

Note: Applies to both reads and writes.

RFI, bit [3], when separate read/write controls

Fault handling interrupt on reads. As the FI bit, except for reads only.

UI, bit [2], when combined read/write control

Uncorrected error recovery interrupt enable. When enabled, the error recovery interrupt is generated for all detected Uncorrected errors that are not deferred. The possible values of this bit are:

`0` Error recovery interrupt disabled.

`1` Error recovery interrupt enabled.

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error. If the node does not support this control, this bit is RES0.

Note: Applies to both reads and writes.

RUI, bit [2], when separate read/write controls

Error recovery interrupt on reads. As the UI bit, except for reads only.

Bit [1]

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

This bit reads as an IMPLEMENTATION DEFINED value and writes to this bit have IMPLEMENTATION DEFINED behavior.

ED, bit [0]

Enable error detection and correction at the node. When disabled, error detection and correction is disabled on reads. ARM recommends that, when disabled, correct error detection and correction codes are written for writes, unless disabled by an IMPLEMENTATION DEFINED control for error injection. The possible values of this bit are:

`0` Error detection and correction disabled.

`1` Error detection and correction enabled.

If the node does not support this control, this bit is RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

This bit resets to zero on a Cold reset.

Note: The bit is set to 0 on Cold reset, meaning errors are not detected or corrected from Cold reset. This allows boot software to initialize a node without signaling errors. When the node is initialized, software can enable error detection.

18.1.3 ERR<*n*>FR, Error Record Feature Register

The ERR<*n*>FR characteristics are:

Purpose

Defines which of the common architecturally-defined features are implemented and, of the implemented features, which are software programmable. If a single node has multiple records, then only the first record has a populated error feature register. If any other record is selected, bits [31:0] of the error feature register are RES0.

Usage constraints

None.

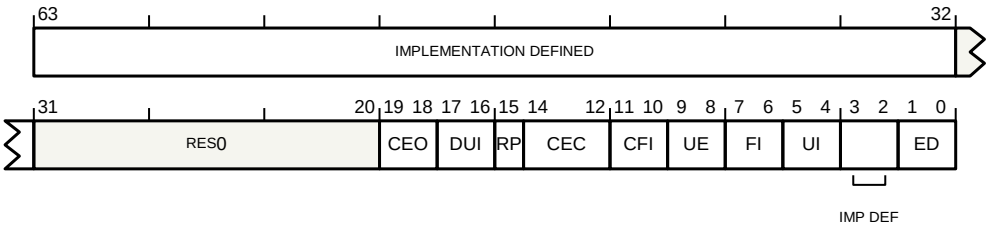
Configurations

Always implemented.

Attributes

64-bit read-only register located at offset `0x000` + 64×*n*.

The ERR<*n*>FR bit assignments are:



Bits [63:32]

Reserved for identifying IMPLEMENTATION DEFINED controls. This field reads as an IMPLEMENTATION DEFINED value.

Bits [31:20]

Reserved. This field is RES0.

CEO, bits [19:18]

Corrected Error overwrite. Indicates the behavior when a second Corrected error is detected after a first Corrected error has been recorded by the node. The defined values of this field are:

`00` Count Corrected error if a counter is implemented. Keep the previous error syndrome. If the counter

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

overflows, or no counter is implemented. ERR<n>STATUS.OF is set to 1.

`01` Count Corrected error. If ERR<n>STATUS.OF == 1 before the Corrected error is counted, keep the previous syndrome. Otherwise the previous syndrome is overwritten. If the counter overflows, ERR<n>STATUS.OF is set to 1.

This field must be `0b00` if no Corrected error counter is implemented. See also Writing the error record.

DUI, bits [17:16]

Error recovery interrupt for deferred errors. Indicates whether the node implements a control for enabling error recovery interrupts on deferred errors. The defined values of this field are:

`00` Does not support feature.

`10` Feature is controllable.

`11` Feature is controllable with independent controls for reads and writes.

All other values are reserved.

If UI == `0b00`, this field is RES0.

RP, bit [15]

Repeat counter. Indicates whether the node implements a repeat Corrected error counter. The defined values of this bit are:

`0` A single CE counter is implemented.

`1` A first (repeat) counter and a second (other) counter are implemented. The repeat counter is the same size as the primary error counter.

If CEC == `0b00`, this bit is RES0.

CEC, bits [14:12]

Corrected Error Counter. Indicates whether the node implements a standard Corrected error counter (CE counter) mechanism in ERR<n>MISC0.. The defined values of this field are:

`000` Does not implement the standard Corrected error counter model.

`010` Implements an 8-bit Corrected error counter in ERR<n>MISC0. [39:32].

`100` Implements a 16-bit Corrected error counter in ERR<n>MISC0. [47:32].

All other values are reserved.

Note: Implementations might include other error counter models, or might include the standard model and not indicate this in ERR<*n*>FR.

CFI, bits [11:10]

Fault handling interrupt for corrected errors. Indicates whether the node implements a control for enabling fault handling interrupts on corrected errors. The defined values of this field are:

`00` Does not support feature.

`10` Feature is controllable.

`11` Feature is controllable with independent controls for reads and writes.

All other values are reserved.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

If FI == `0b00`, this field is RES0.

UE, bits [9:8]

In-band uncorrected error reporting. Indicates whether the node implements in-band uncorrected error reporting (external aborts), and, if so, whether it implements controls for enabling and disabling it. The defined values of this field are:

`00` Does not support feature.

`01` Feature always enabled.

`10` Feature is controllable.

`11` Feature is controllable with independent controls for reads and writes.

FI, bits [7:6]

Fault handling interrupt. Indicates whether the node implements a fault handling interrupt, and, if so, whether it implements controls for enabling and disabling it. The defined values of this field are:

`00` Does not support feature.

`01` Feature always enabled.

`10` Feature is controllable.

`11` Feature is controllable with independent controls for reads and writes.

UI, bits [5:4]

Error recovery interrupt for uncorrected errors. Indicates whether the node implements an error recovery interrupt, and, if so, whether it implements controls for enabling and disabling it. The defined values of this field are:

`00` Does not support feature.

`01` Feature always enabled.

`10` Feature is controllable.

`11` Feature is controllable with independent controls for reads and writes.

Bits [3:2]

This field reads as an IMPLEMENTATION DEFINED value.

ED, bits [1:0]

Error detection and correction. Indicates whether the node implements controls for enabling and disabling error detection and, if implemented, correction. The defined values of this field are:

`01` Feature always enabled.

`10` Feature is controllable.

All other values are reserved.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

18.1.4 ERR<*n*>MISC0, Error Record Miscellaneous Register 0

The ERR<*n*>MISC0 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers contain:

- Corrected error counter(s), if the node supports the counting of Corrected errors.
- Information to identify the Field Replaceable Unit (FRU) in which the error was detected, and might contain enough information to locate error within that FRU.
- Other state information not present in the corresponding status and address registers.

If the node supports the architecturally-defined error counter, this is implemented in ERR<*n*>MISC0.

Usage constraints

If ERR<n>STATUS.MV is set to 1, it is IMPLEMENTATION DEFINED whether fields of ERR<*n*>MISC0 ignores writes. ARM recommends that miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write. This allows a counter to be reset in the presence of a persistent error. Miscellaneous syndrome for the most recently recorded error, such as information locating a FRU for that error, should ignore writes. This prevents information being lost if an error is detected whilst the previous error is being logged.

Configurations

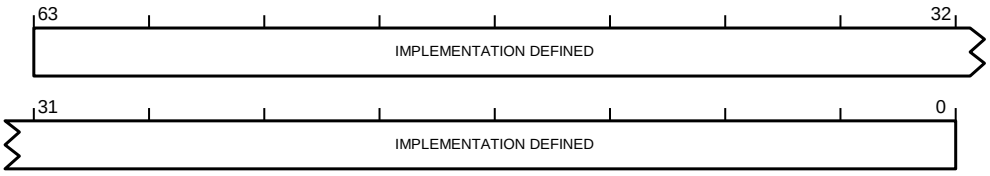
Always implemented.

Attributes

64-bit read/write register located at offset `0x020` + `64`×*n*.

18.1.4.1 ERR<n>MISC0 (contents are IMPLEMENTATION DEFINED)

The ERR<*n*>MISC0 (contents are IMPLEMENTATION DEFINED) bit assignments are:



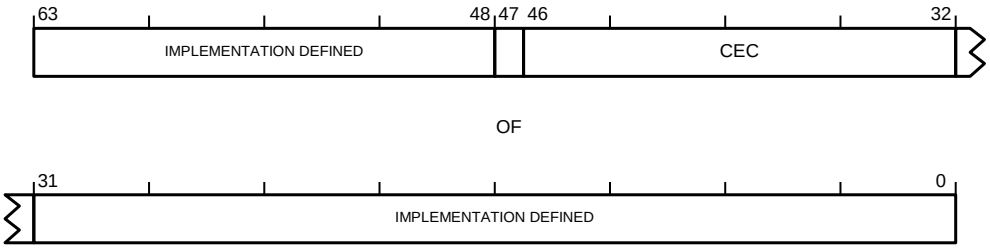
Bits [63:0], when contents are IMPLEMENTATION DEFINED

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

18.1.4.2 ERR<n>MISC0 (standard 16-bit CE counter)

The ERR<*n*>MISC0 (standard 16-bit CE counter) bit assignments are:



Bits [63:48,31:0], when standard 16-bit CE counter

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

OF, bit [47], when standard 16-bit CE counter

Sticky overflow bit. The possible values of this bit are:

`0` Counter has not overflowed.

`1` Counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and the overflow bit is set to 1.

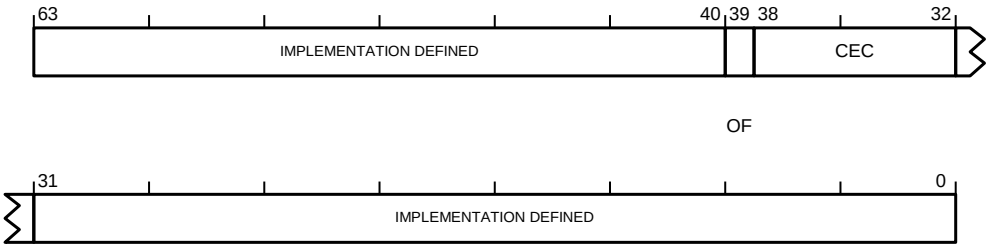
CEC, bits [46:32], when standard 16-bit CE counter

Corrected error count.

This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

18.1.4.3 ERR<n>MISC0 (standard 8-bit CE counter)

The ERR<*n*>MISC0 (standard 8-bit CE counter) bit assignments are:



Bits [63:40,31:0], when standard 8-bit CE counter

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

OF, bit [39], when standard 8-bit CE counter

Sticky overflow bit. The possible values of this bit are:

`0` Counter has not overflowed.

`1` Counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and the overflow bit is set to 1.

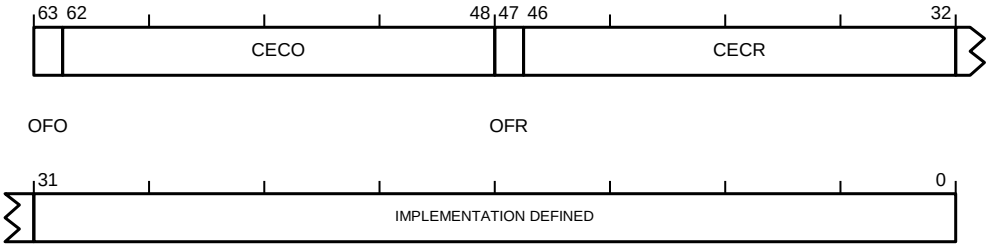
CEC, bits [38:32], when standard 8-bit CE counter

Corrected error count.

This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

18.1.4.4 ERR<n>MISC0 (standard 16-bit CE counter pair)

The ERR<*n*>MISC0 (standard 16-bit CE counter pair) bit assignments are:



OFO, bit [63], when standard 16-bit CE counter pair

Sticky overflow bit, other. The possible values of this bit are:

`0` Other counter has not overflowed.

`1` Other counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

CECO, bits [62:48], when standard 16-bit CE counter pair

Corrected error count, other. Incremented for each Corrected error that does not match the recorded syndrome.

This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

OFR, bit [47], when standard 16-bit CE counter pair

Sticky overflow bit, repeat. The possible values of this bit are:

`0` Repeat counter has not overflowed.

`1` Repeat counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

CECR, bits [46:32], when standard 16-bit CE counter pair

Corrected error count, repeat. Incremented for the first recorded error, which also records other syndrome, and then again for each Corrected error that matches the recorded syndrome.

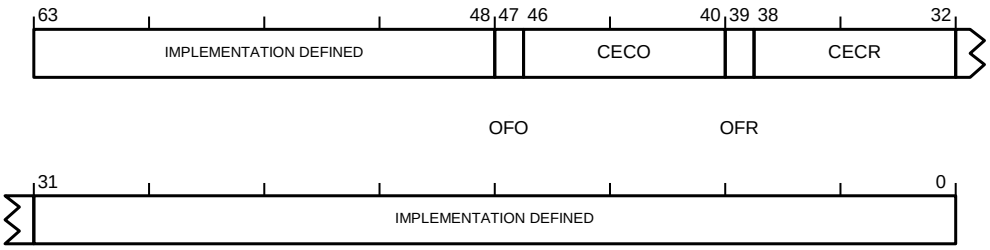
This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

Bits [31:0], when standard 16-bit CE counter pair

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

18.1.4.5 ERR<n>MISC0 (standard 8-bit CE counter pair)

The ERR<*n*>MISC0 (standard 8-bit CE counter pair) bit assignments are:



Bits [63:48,31:0], when standard 8-bit CE counter pair

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

OFO, bit [47], when standard 8-bit CE counter pair

Sticky overflow bit, other. The possible values of this bit are:

`0` Other counter has not overflowed.

`1` Other counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

CECO, bits [46:40], when standard 8-bit CE counter pair

Corrected error count, other. Incremented for each Corrected error that does not match the recorded syndrome.

This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

OFR, bit [39], when standard 8-bit CE counter pair

Sticky overflow bit, repeat. The possible values of this bit are:

`0` Repeat counter has not overflowed.

`1` Repeat counter has overflowed.

The fault handling interrupt is generated when the corrected fault handling interrupt is enabled and either overflow bit is set to 1.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

CECR, bits [38:32], when standard 8-bit CE counter pair

Corrected error count, repeat. Incremented for the first recorded error, which also records other syndrome, and then again for each Corrected error that matches the recorded syndrome.

This field resets to an IMPLEMENTATION DEFINED which might be UNKNOWN on a Cold reset. If the reset value is UNKNOWN, then the value of this field remains UNKNOWN until software initializes it.

18.1.5 ERR<*n*>MISC1, Error Record Miscellaneous Register 1

The ERR<*n*>MISC1 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers contain:

- Corrected error counter(s), if the node supports the counting of Corrected errors.
- Information to identify the Field Replaceable Unit (FRU) in which the error was detected, and might contain enough information to locate error within that FRU.
- Other state information not present in the corresponding status and address registers.

Usage constraints

If ERR<n>STATUS.MV is set to 1, it is IMPLEMENTATION DEFINED whether fields of ERR<*n*>MISC1 ignores writes. ARM recommends that miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write. This allows a counter to be reset in the presence of a persistent error. Miscellaneous syndrome for the most recently recorded error, such as information locating a FRU for that error, should ignore writes. This prevents information being lost if an error is detected whilst the previous error is being logged.

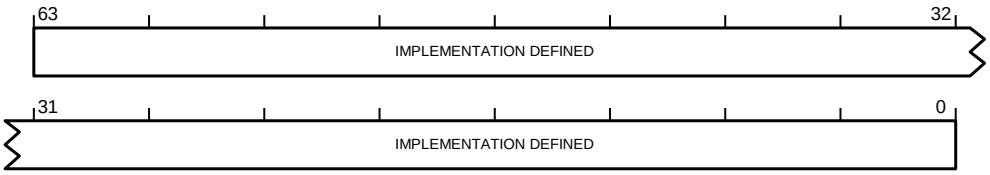
Configurations

Always implemented.

Attributes

64-bit read/write register located at offset `0x028` + `64`×*n*.

The ERR<*n*>MISC1 bit assignments are:



Bits [63:0]

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

18.1.6 ERR<*n*>STATUS, Error Record Primary Status Register

The ERR<*n*>STATUS characteristics are:

Purpose

Contains the following information for the error record:

- Whether any error has been detected (valid).
- Whether any detected error was not corrected, and returned to a master.
- Whether any detected error was not corrected and deferred.
- Whether a second error of the same type was detected before the first error was handled by software (overflow).
- Whether any error has been reported.
- Whether the other error record register(s) contain valid information.
- Whether the error was due to poison data or detected by an EDC.
- A primary error code.
- An IMPLEMENTATION DEFINED extended error code.

Within this register:

- The {AV, V, MV} bits are valid bits that define whether the error record registers are valid.
- The {UE, OF, CE, DE, UET} bits encode the type of error(s) recorded.
- The {ER, PN, IERR, SERR} fields are syndrome fields.

Usage constraints

After reading the status register, software must clear the valid bits to allow new errors to be recorded. Between reading the register and clearing the valid bits, a new error might have overwritten the register. To prevent this error being lost, some control bits use a form of R/W1C.

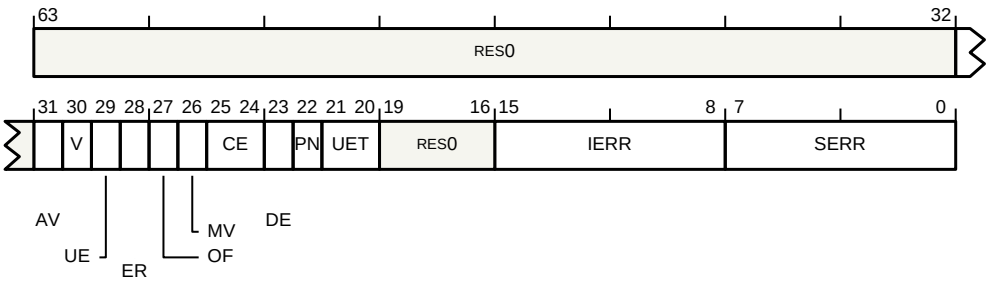
Configurations

Always implemented.

Attributes

64-bit read/write register located at offset `0x010` + `64`×*n*.

The ERR<*n*>STATUS bit assignments are:



Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Bits [63:32,19:16]

Reserved. This field is RES0.

AV, bit [31]

Address Valid. The possible values of this bit are:

`0` ERR<n>ADDR not valid.

`1` ERR<n>ADDR contains an address associated with the highest priority error recorded by this record.

Direct writes to this bit are ignored if any of the CE, DE, or UE bits is set to 1, and the highest priority of these is not being cleared to 0 in the same write. This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset.

V, bit [30]

Status Register valid. The possible values of this bit are:

`0` ERR<*n*>STATUS not valid.

`1` ERR<*n*>STATUS valid. At least one error has been recorded.

Direct writes to this bit are ignored if any of the UE, DE, or CE bits is set to 1 and is not being cleared to 0 in the same write. This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset.

UE, bit [29]

Uncorrected error(s). The possible values of this bit are:

`0` No errors that could neither be corrected nor deferred.

`1` At least one error detected that has neither been corrected nor deferred.

Direct writes to this bit are ignored if the OF bit is set to 1 and is not being cleared to zero in the same write. This bit is read/write-one-to-clear.

ER, bit [28]

Error Reported. The possible values of this bit are:

`0` No in-band error (external abort) reported.

`1` An external abort was signaled by the node to the master making the access or other transaction. This can be because any of:

- The applicable one of the ERR<n>CTLR.{WUE,RUE,UE} bits is implemented and was set to 1 when an Uncorrected error was detected.
- The applicable one of the ERR<n>CTLR.{WUE,RUE,UE} bits is not implemented and the node always reports errors.

Direct writes to this bit are ignored if any of the CE, DE, or UE bits is set to 1, and the highest priority of these is not being cleared to 0 in the same write.

RES0 for a Corrected error. It is IMPLEMENTATION DEFINED whether this bit can be set for a Deferred error.

This bit is read/write-one-to-clear.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: An external abort signaled by the node might be masked and not generate any exception.

OF, bit [27]

Overflow.

Multiple errors detected. This bit is set to 1 when:

- An Uncorrected error is detected and the previous error syndrome is kept because UE == 1.
- A Deferred error is detected and the previous error syndrome is kept is discarded because DE == 1.
- A Corrected error is detected and either:
  - A Corrected error counter is implemented, and the counter overflows.
  - No Corrected error counter is implemented, and CE ≠ `0b00`. The CE field might be updated for the new Corrected error.

It is IMPLEMENTATION DEFINED whether the bit is set to 1 when:

- A Deferred error is detected and UE == 1.
- A Corrected error is detected and either or both the DE or UE bits are set to 1, and either the Corrected error counter overflows or is not implemented.

It is IMPLEMENTATION DEFINED whether this bit is reset to 0 when an error is detected and the previous error syndrome is overwritten.

The possible values of this bit are:

`0`
- If UE == 1, then no error syndrome for an Uncorrected error has been discarded.
- If UE == 0 and DE == 1, then no error syndrome for a Deferred error has been discarded.
- If UE == 0, DE == 0 and CE ≠ `0b00` then:
  - If a Corrected error counter is implemented, it has not overflowed.
  - If no Corrected error counter is implemented, no error syndrome for a Corrected error has been discarded.

Note: An error syndrome might have been discarded but not recorded in this bit because a higher priority syndrome is recorded or has overwritten this bit.

`1` At least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed.

For more information, see Writing the error record. This bit is read/write-one-to-clear.

MV, bit [26]

Miscellaneous Registers Valid. The possible values of this bit are:

`0` ERR<n>MISC0 and ERR<n>MISC1 not valid.

`1` The IMPLEMENTATION DEFINED contents of the ERR<n>MISC0 and ERR<n>MISC1 registers contains additional information for an error recorded by this record.

Direct writes to this bit are ignored if any of the CE, DE, or UE bits is set to 1, and the highest priority of these is not being cleared to 0 in the same write. This bit is read/write-one-to-clear.

This bit resets to zero on a Cold reset.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Note: If the ERR<n>MISC0 and ERR<n>MISC1 registers can contain additional information for a previously recorded error, then the contents must be self-describing to software or a user. For example, certain fields might relate only to Corrected errors, and other fields only to the most recently error that was not discarded.

CE, bits [25:24]

Corrected error(s). The possible values of this field are:

`00` No errors were corrected.

`01` At least one transient error was corrected.

`10` At least one error was corrected.

`11` At least one persistent error was corrected.

The mechanism by which a node detects whether a correctable error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, the node sets this field to `0b10` when an error is corrected.

Direct writes to this field are ignored if the OF bit is set to 1 and is not being cleared to 0 in the same write.

This field is read/write-one-to-clear.

DE, bit [23]

Deferred error(s). The possible values of this bit are:

`0` No errors were deferred.

`1` At least one error was not corrected and deferred.

Support for deferring errors is IMPLEMENTATION DEFINED.

Direct writes to this bit are ignored if the OF bit is set to 1 and is not being cleared to 0 in the same write.

This bit is read/write-one-to-clear.

PN, bit [22]

Poison. The possible values of this bit are:

`0` Uncorrected or deferred error from a corrupted value.

Note: If a node detects a corrupted value and defers the error by *producing* poison, then this bit is set to 0 at the producer node.

`1` Uncorrected error or Deferred error from a poisoned value. Indicates that an error was due to detecting a poison value rather than detecting a corrupted value.

Note: This might only be an indication of poison, as in some EDC schemes, a poisoned value is encoded as an unlikely form of corrupted data, meaning it is possible to mistake a corrupted value as a poisoned value.

It is IMPLEMENTATION DEFINED whether a node can distinguish a poisoned value from a corrupted value.

Valid only if at least one of the UE or DE bits are set. Otherwise RES0.

Direct writes to this bit are ignored if any of the CE, DE, or UE bits is set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This bit is read/write-one-to-clear.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

UET, bits [21:20]

Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error. The possible values of this field are:

`0b10` Uncorrected error, Latent or Restartable error (UEO).

`0b11` Uncorrected error, Signaled or Recoverable error (UER).

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

`0b00` Uncorrected error, Uncontainable error (UC).

Valid only if the UE bit is set. Otherwise RES0.

Direct writes to this field are ignored if any of the CE, DE, or UE bits is set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This field is read/write-one-to-clear.

Note: Software might use the information in the error record registers to determine what recovery is necessary.

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code SERR value. Further IMPLEMENTATION DEFINED information can be placed in the MISC registers.

Direct writes to this field are ignored if any of the CE, DE, or UE bits is set to 1, and the highest priority of these is not being cleared to 0 in the same write.

The set of values that IERR can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, the value read back from IERR is UNKNOWN.

Note: This means that an implementation can choose to implement one or more bits of IERR as fixed read-as-zero or read-as-one values.

SERR, bits [7:0]

Architecturally-defined primary error code. Indicates the type of error. The primary error code might be used by a fault handling agent to triage an error without requiring device-specific code. For example, to count and threshold corrected errors in software, or generate a short log entry. The possible values of this field are:

`0` No error.

`1` IMPLEMENTATION DEFINED error.

`2` Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.

`3` IMPLEMENTATION DEFINED pin. For example, **nSEI** pin.

`4` Assertion failure. For example, consistency failure.

`5` Internal data path. For example, parity on ALU result.

`6` Data value from associative memory. For example, ECC error on cache data.

`7` Address/control value(s) from associative memory. For example, ECC error on cache tag.

`8` Data value from a TLB. For example, ECC error on TLB data.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

`9` Address/control value(s) from a TLB. For example, ECC error on TLB tag.

`10` Data value from producer. For example, parity error on write data bus.

`11` Address/control value(s) from producer. For example, parity error on address bus.

`12` Data value from (non-associative) external memory. For example, ECC error in SDRAM.

`13` Illegal address (software fault). For example, access to unpopulated memory.

`14` Illegal access (software fault). For example, byte write to word register.

`15` Illegal state (software fault). For example, device not ready.

`16` Internal data register. For example, parity on a FP&SIMD register. For a PE, all general-purpose, stack pointer, and FP&SIMD registers are data registers.

`17` Internal control register. For example, Parity on a system register. For a PE, all registers other than general-purpose, stack pointer, and FP&SIMD registers are control registers.

`18` Error response from slave. For example, error response from cache write-back.

`19` External timeout. For example, timeout on interaction with another node.

`20` Internal timeout. For example, timeout on interface within the node.

All other values are reserved.

Direct writes to this field are ignored if any of the CE, DE, or UE bits is set to 1, and the highest priority of these is not being cleared to 0 in the same write.

The subset of architecturally-defined values that SERR can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, the value read back from SERR is UNKNOWN.

Note: This means that an implementation can choose to implement one or more bits of SERR as fixed read-as-zero or read-as-one values.

18.1.7 ERRDEVARCH, Device Architecture Register

The ERRDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the component. This allows software to make some use of a component that conforms to an architecture definition but whose part number is not necessarily recognized, or to disambiguate multiple architecturally-defined sub-components of a system.

Usage constraints

None.

Configurations

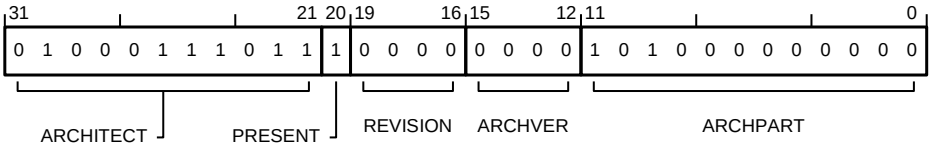
Always implemented.

Attributes

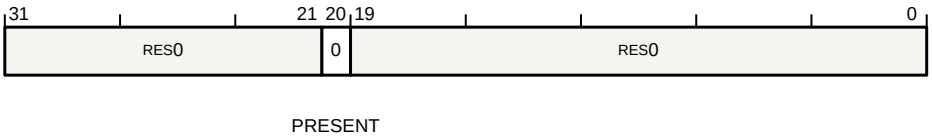
32-bit read-only register located at offset `0xFBC`.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

When DEVARCH is implemented, the ERRDEVARCH bit assignments are:



When DEVARCH is not implemented, the ERRDEVARCH bit assignments are:



Bits [31:21,19:0], when DEVARCH is not implemented

Reserved. This field is RES0.

ARCHITECT, bits [31:21], when DEVARCH is implemented

Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code. The defined values of this field are:

`0x23B` JEP106 continuation code `0x4`, ID code `0x3B`. ARM Limited.

Other values are defined by the JEDEC JEP106 standard. This field reads as `0x23B`.

PRESENT, bit [20]

Defines that the DEVARCH register is present. The defined values of this bit are:

`0` DEVARCH information not present. The rest of the register is RES0.

`1` DEVARCH information present.

REVISION, bits [19:16], when DEVARCH is implemented

Defines the architecture revision. The defined values of this field are:

`0000` RAS system architecture v1.0.

This field reads as `0b0000`.

ARCHVER, bits [15:12], when DEVARCH is implemented

Defines the architecture version of the component. The defined values of this field are:

`0000` RAS system architecture v1.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, such that ARCHVER is ARCHID[15:12]. This field reads as `0b0000`.

ARCHPART, bits [11:0], when DEVARCH is implemented

Defines the architecture of the component. The defined values of this field are:

`0xA00` RAS system architecture.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

ARCHVER and ARCHPART are also defined as a single field, ARCHID, such that ARCHPART is ARCHID[11:0]. This field reads as `0xA00`.

18.1.8 ERRGSR<*n*>, Error Group Status Register

The ERRGSR<*n*> characteristics are:

Purpose

Each ERRGSR<*n*> register shows the status for up to 64 records in the group. <*n*> selects the set of 64 records from the records in the group.

Usage constraints

None.

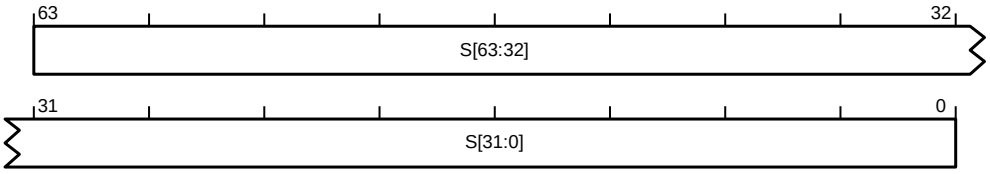
Configurations

Always implemented.

Attributes

64-bit read-only register located at offset `0xE00` + 8×*n*.

The ERRGSR<*n*> bit assignments are:



S, bits [63:0]

Bit <*q*> maps to S[*m*], where *m* = 64×*n* + *q*, the status for Error Record <*m*>. A read-only copy of ERR<*m*>STATUS.V. The defined values of this field are:

`0` No error.

`1` One or more errors.

If <*m*> is greater than or equal to the number of implemented records, or the record does not support this type of reporting, the bit is RES0.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

18.1.9 ERRIDR, Error Record ID Register

The ERRIDR characteristics are:

Purpose

Defines the number of error records in this group.

Usage constraints

None.

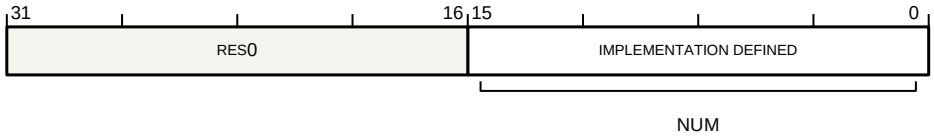
Configurations

Always implemented.

Attributes

32-bit read-only register located at offset `0xFC8`.

The ERRIDR bit assignments are:



Bits [31:16]

Reserved. This field is RES0.

NUM, bits [15:0]

Number of records in this group. Each record is notionally owned by a node. A node might own multiple records.

This field reads as an IMPLEMENTATION DEFINED value.

18.1.10 ERRIRQCR<*n*>, Error Interrupt Configuration Register

The ERRIRQCR<0-14> characteristics are:

Purpose

The ERRIRQCR<*n*> registers are reserved for IMPLEMENTATION DEFINED interrupt configuration registers.

Usage constraints

None.

Configurations

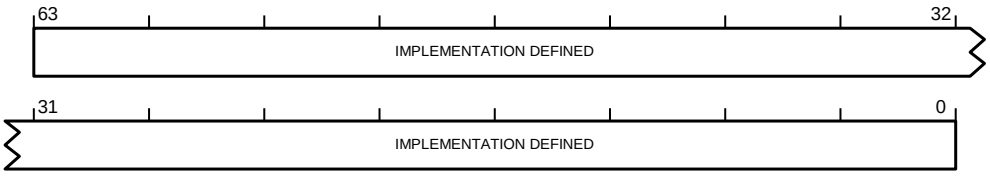
Present only if interrupt configuration registers are implemented. RES0 otherwise.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential

Attributes

64-bit read/write register located at offset `0xE80` + 8×*n*.

The ERRIRQCR<0-14> bit assignments are:



Bits [63:0]

IMPLEMENTATION DEFINED controls. The content of these registers is IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Note: For example, in a system supporting GIC message signaled interrupts, these registers might be used to configure the address of and payload for a write to a GICD_SETSPI register.

Copyright © 2016-2017 ARM or its affiliates. All rights reserved.
Non-Confidential