Learn the architecture - Realm Management Extension

Version 1.2

Guide

Non-Confidential

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates).
All rights reserved.

Issue 01
den0126_0102_01_en

Learn the architecture - Realm Management Extension
Guide

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0102-01	26 September 2025	Non-Confidential	Added RME-DA and MEC extensions
0101-02	12 February 2025	Non-Confidential	Minor corrections
0101-01	2 October 2024	Non-Confidential	2024 extensions update
0100-02	23 June 2021	Non-Confidential	First release

Proprietary Notice

This document is protected by copyright and other related rights and the use 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 the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. 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, any patents, copyrights, trade secrets, trademarks, or other rights.

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.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted 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.

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 of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. Please follow Arm's trademark usage guidelines at https://www.arm.com/company/policies/trademarks. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Feedback

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

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

Inclusive language commitment

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

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Contents

1. Overview... **9**

2. Security states... **11**

2.1 Controlling the current security state.. 12

2.2 Moving between security states... 13

3. Physical Addresses.. **15**

3.1 Virtual address spaces.. 16

3.2 Controlling output PAS.. 17

3.2.1 Root state translation regimes.. 18

3.2.2 Realm state translation regimes... 19

3.2.3 Secure state translation regimes... 20

3.2.4 Non-secure state translation regimes.. 21

3.2.5 Impact on Translation Lookaside Buffers and caches.. 22

4. Granule Protection Checks.. **23**

4.1 GPTs.. 28

4.2 GPC faults... 31

4.2.1 Granule Protection Faults (GPFs)... 32

4.3 Transitioning a granule between physical address spaces.. 32

4.3.1 Impact on caches.. 33

4.3.2 Impact on TLBs.. 34

4.4 GPC bypass windows.. 34

5. System architecture... **36**

5.1 Main memory protection.. 36

5.2 MPAM.. 37

5.3 RAS.. 37

5.4 Granule Data Isolation (GDI).. 38

5.5 Explicit PAS control for CXL-TSP.. 39

6. SMMU architecture... **40**

6.1 SMMUs in an RME-enabled system.. 40

6.2 Changes to the SMMU architecture for RME.. 42

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.3 Changes to the SMMU architecture for RME DA........ 42
6.4 SMMU support for GPC........ 43
6.4.1 How GPC is controlled in an SMMU........ 43
6.4.2 Invalidation of GPT information........ 44
6.4.3 GPC faults........ 44
6.4.4 GPC interrupts........ 45
6.4.5 GPC for NoStreamID devices........ 45
6.5 Support for Realm programming interface........ 45
6.5.1 Translation process overview........ 45
6.5.2 Stream security........ 46
6.5.3 StreamWorld changes........ 47
6.5.4 Output physical address space for a Realm transaction........ 47
6.5.5 Data structures changes........ 48
6.5.6 Realm Register Pages........ 48
6.5.7 Interrupts........ 48
6.6 DPT........ 49
6.6.1 Split-stage ATS versus Full ATS with DPT........ 49
6.6.2 DPT lookup........ 51
6.6.3 DPT descriptors........ 52
6.6.4 DPT caching........ 52
6.6.5 DPT lookup errors........ 53
6.7 Memory System Resource Partitioning and Monitoring (MPAM)........ 54
6.8 Performance Monitoring Counter Group enhancements........ 54
6.8.1 Counting events related to DPT lookups........ 55
6.8.2 StreamID and Security State Filtering........ 55
6.8.3 MPAM PARTID and PMG filtering........ 56
6.8.4 Counting of non-attributable events........ 56

7. Memory Encryption Contexts extension........ 57
7.1 MECIDs........ 57
7.2 MECID allocation........ 57
7.2.1 Effect of MEC on PAS........ 57
7.3 MECID width........ 58
7.4 MECID mismatch........ 58
7.5 Memory protection engine (MPE)........ 58
7.6 Memory encryption block size........ 59

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7.7 MECID on SMMU for RME DA.......59

8. SMMU implementation.......60
8.1 Stream security.......60
8.1.1 ACE-Lite.......60
8.1.2 DTU-TBU.......61
8.1.3 DTI-ATS.......61
8.1.4 LTI.......61
8.2 PAS encoding.......62
8.2.1 ACE-Lite.......62
8.2.2 DTI-TBU.......62
8.2.3 DTI-ATS.......63
8.2.4 LTI.......63
8.3 NoStreamID accesses.......63
8.3.1 ACE-Lite.......63
8.3.2 LTI.......64
8.4 Memory System Resource Partitioning and Monitoring.......64
8.4.1 MPAM in ACE-Lite.......64
8.4.2 MPAM in DTI-TBU.......65
8.4.3 MPAM in DTI-ATS.......65
8.4.4 MPAM in LTI.......65
8.5 Memory Encryption Context Identifier.......66
8.5.1 ACE-Lite.......66
8.5.2 DTI-TBU.......66
8.5.3 DTI-ATS.......67
8.5.4 LTI.......67
8.6 PCIe IDE interactions.......67
8.6.1 TDISP XT Extensions.......68

9. Debug, trace, and profiling.......69
9.1 External debug.......69
9.2 Self-hosted debug.......71
9.3 Self-hosted trace and SPE.......71
9.4 Performance monitoring.......72
9.5 Branch Record Buffer Extension.......72

10. Related information.......73

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

11. Next steps..74

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

1. Overview

This guide introduces the Realm Management Extension (RME), an extension to the Armv9-A architecture.

RME is the hardware component of the Arm Confidential Compute Architecture (Arm CCA), which combines hardware and software elements. RME enables the dynamic transfer of resources and memory to a new protected address space that higher-privileged software or TrustZone firmware cannot access. Arm CCA uses this address space to construct protected execution environments called Realms. Refer to Introducing Arm Confidential Compute Architecture.

Realms enable lower-privileged software, such as an application or a Virtual Machine (VM), to protect its content. Higher-privileged software is still responsible for allocating and managing the resources that a Realm uses. However, the higher-privileged software cannot access the Realm's contents or interfere with its execution flow. With RME, the memory available to TrustZone Software entities can change dynamically.

This guide describes the key hardware features that RME introduces. You will learn about the following concepts:

- New security states and Physical Address (PA) spaces introduced by RME
- Memory reassignment between PA spaces
- System requirements for RME-enabled systems

This guide explains the following changes that RME introduces to the processor architecture:

- Additional Security states
- Additional Physical addresses
- Support for Granule Protection Checks, which enable granules of memory to be dynamically assigned to a physical address space

---

Diversity and inclusion are important values to Arm. Because of this, we are re-evaluating the terminology we use in our documentation. Older Arm documentation uses the terms *master* and *slave*.

This guide uses replacement terminology, as follows:

- The term *requester* replaces *master* in older documentation
- The term *subordinate* replaces *slave* in older documentation

Before you begin

You should be familiar with the AArch64 Exception model, memory management, and TrustZone. To learn more about these topics, see the following guides:

- AArch64 Exception model introduces the exception and privilege model in AArch64

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- AArch64 memory management introduces the MMU, which is used to control virtual to physical address translation
- TrustZone for AArch64 introduces TrustZone, an efficient, system-side approach to security with hardware-enforced isolation built into the CPU

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

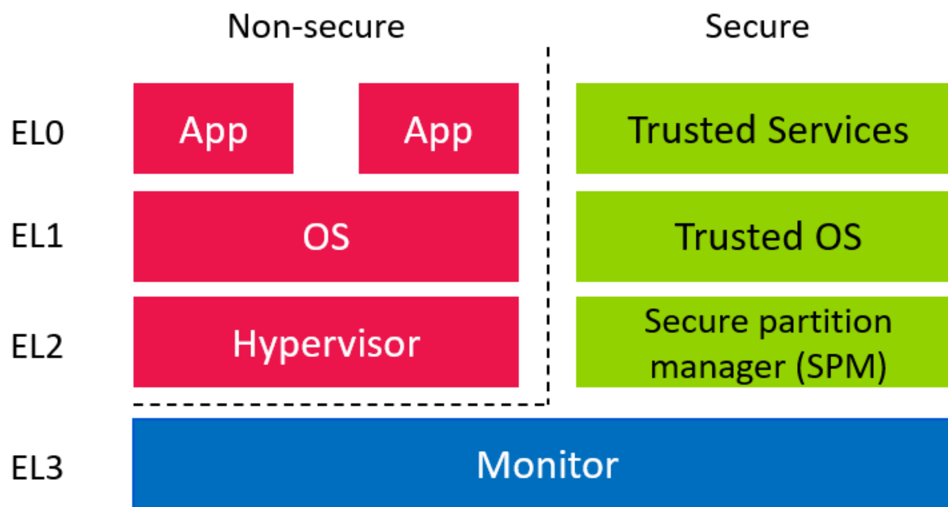
2. Security states

RME builds on Arm TrustZone technology. TrustZone, introduced in Armv6, defines two security states:

- Secure state
- Non-secure state

The following diagram shows these two security states in AArch64 and the software components typically associated with each state:

Figure 2-1: Security states before RME



The architecture isolates software in Secure state from software in Non-secure state. This isolation enables a software architecture in which trusted code runs in Secure state and is protected from code in Non-secure state.

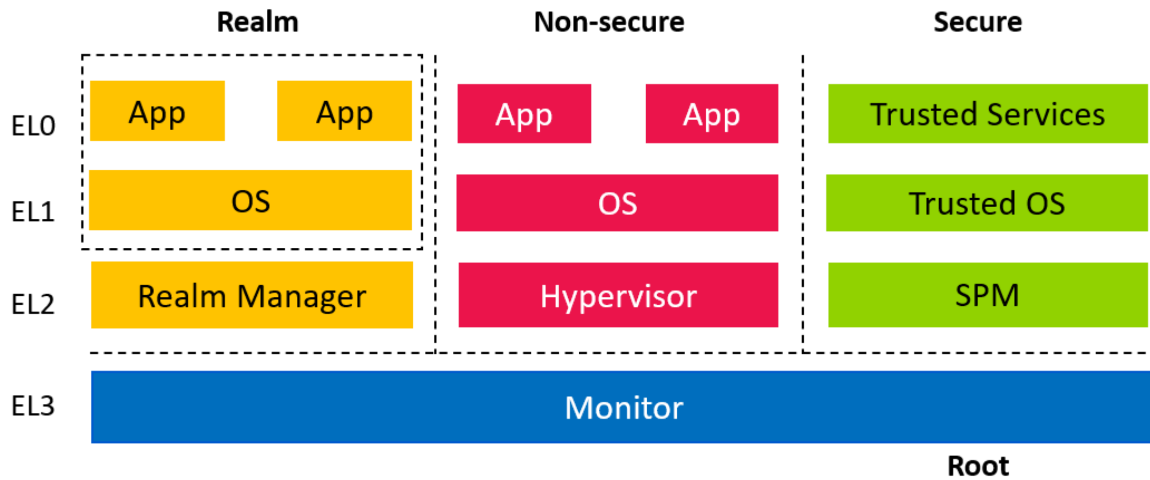
RME extends this model and introduces four security states:

- Secure state
- Non-secure state
- Realm state
- Root state

The following diagram shows the security states in an RME-enabled Processing Element (PE) and how these states map to exception levels (ELx):

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 2-2: Security states with RME



Maintaining Secure state ensures backwards compatibility with existing TrustZone use cases. These use cases can also be upgraded to take advantage of RME features, such as dynamic memory assignment.

In Realm state, protected environments called Realms can be constructed. RME extends the isolation model introduced by TrustZone.

The architecture provides the following isolation guarantees:

- Secure state is isolated from Non-secure and Realm states
- Realm state is isolated from Non-secure and Secure states

This isolation model supports a software architecture in which software in Secure and Realm states are mutually distrusting.

With RME, EL3 is moved out of Secure state into its own security state called Root. RME isolates EL3 from all other security states. EL3 hosts the platform and initial boot code and must be trusted by software in Non-secure, Secure, and Realm states. Because these states do not trust each other, EL3 requires a distinct security state.

2.1 Controlling the current security state

A combination of the exception level and the SCR_EL3 register controls the current security state.

EL3 now operates in its own Root security state. While in EL3, the security state is always Root, and no other exception level can enter Root state.

At lower exception levels (EL0, EL1, and EL2), the `NS` and `NSE` fields in `SCR_EL3` control the current security state. The valid combinations are shown in the following table:

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

SCR_EL3.{NSE,NS}	Security state
{0,0}	Secure
{0,1}	Non-secure
{1,0}	-
{1,1}	Realm

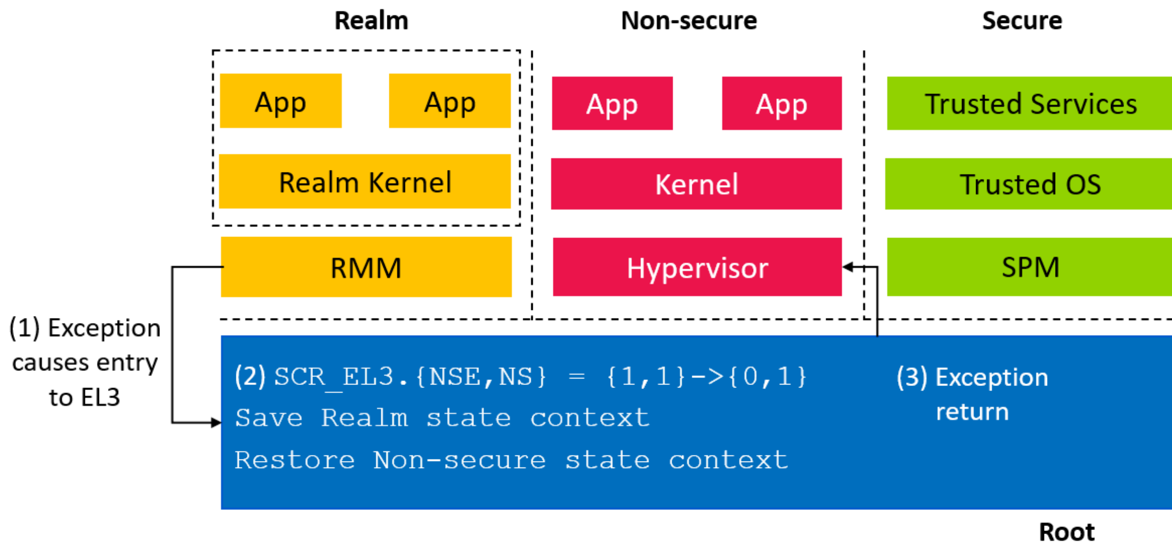
There is no encoding for Root state. While executing in EL3, the security state is always reported as Root, regardless of the values in `SCR_EL3.{NSE,NS}`.

In EL3, the current `SCR_EL3.{NSE,NS}` value controls certain operations. For example, when a Translation Lookaside Buffer (TLB) invalidation instruction is issued at EL3 for a lower exception level, the `SCR_EL3.{NSE,NS}` fields determine which security state the operation targets.

2.2 Moving between security states

The mechanism for transitioning between security states is inherited from TrustZone. To change the security state, execution must pass through EL3, as shown in the following diagram:

Figure 2-3: Changing security state example



The diagram shows the following steps:

1. Execution begins in Realm state with `SCR_EL3.{NSE,NS}` set to `{1,1}`. The software issues a Secure Monitor Call (SMC) instruction, which triggers an exception to EL3.
2. The processor enters EL3 and transitions to Root state, as EL3 always operates in Root state. The `SCR_EL3.{NSE,NS}` value reflects the originating security state. The EL3 software changes `SCR_EL3.{NSE,NS}` to `{0,1}`, which corresponds to Non-secure state, and executes an Exception Return (ERET) instruction.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. The ERET instruction exits EL3. On exit, the `SCR_EL3.{NSE,NS}` setting determines the new security state—Non-secure in this case.

There is a single copy of the vector registers, general-purpose registers, and most system registers. When transitioning between security states, it is the responsibility of the EL3 software to save and restore register context. This context management software is called the Monitor.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3. Physical Addresses

In addition to two Security states, TrustZone provides the following two Physical Address Spaces (PAS):

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

These separate PA spaces form part of the TrustZone isolation guarantee. Secure state can access the address in both Secure PA space and Non-secure PA space. However, Non-secure state cannot access an address in a Secure PA space. This isolation means that there are confidentiality and integrity guarantees for data belonging to Secure state.

RME extends this guarantee to support the following PA spaces:

- Secure physical address space
- Non-secure physical address space
- Realm physical address space
- Root physical address space

The architecture limits which PA spaces are accessible in each Security state. The following table shows the PA spaces that are accessible in each Security state:

Physical address space (PAS)	Secure state	Non-secure state	Realm state	Root state
Secure PAS	Y	N	N	Y
Non-secure PAS	Y	Y	Y	Y
Realm PAS	N	N	Y	Y
Root PAS	N	N	N	Y

In this table, Y means accessible and N means not accessible.

When documentation refers to a physical address, prefixes identify which address space is being referred to, for example:

- SP:`0x8000` means address `0x8000` in the Secure PA space
- NSP:`0x8000` means address `0x8000` in the Non-secure PA space
- RLP:`0x8000` means address `0x8000` in the Realm PA space
- RTP:`0x8000` means address `0x8000` in the Root PA space

Architecturally, each example is an independent memory location. This means that SP:`0x8000` and RTP:`0x8000` are treated as different physical locations. All four locations can exist in an RME-enabled system although in practice, this is unlikely.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

FEAT_RME_GPC2 introduced in Armv9.5-A, extended this model further by enabling some Non-secure PAS regions to be protected. With this feature, the protected *Non-secure only* regions are not accessible to Secure or Realm state.

FEAT_RME_GDI introduced in Armv9.6-A, introduces two new PA spaces in a system called as *System Agent* PA space and *Non-Secure Protected* PA space. For more details, Refer section Granule Data Isolation (GDI).

3.1 Virtual address spaces

RME introduces the following translation regimes for Realm state:

- Realm EL1 and EL0 translation regime:

  This regime includes two virtual address (VA) regions, similar to the Non-secure EL1 and EL0 translation regime. This translation regime is subject to stage 2 translation.
- Realm EL2 and EL0 translation regime:

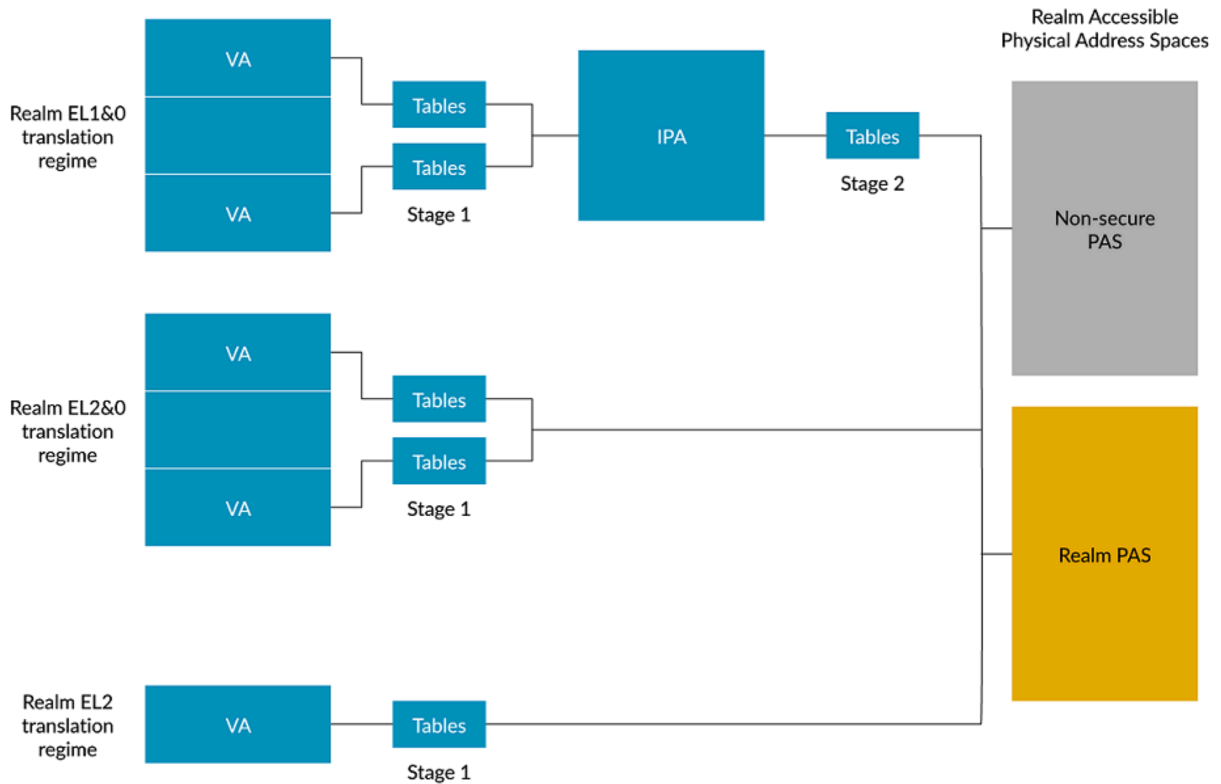
  This regime includes two VA regions, similar to the Secure EL2 and EL0 translation regime.
- Realm EL2 translation regime:

  This regime includes a single VA region, similar to the Secure EL2 translation regime.

The following diagram shows the Realm state translation regimes:

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 3-1: Realm translation regimes



For all Realm translation regimes,table walk(s) can only access Realm PAS. Also, for all Realm translation regimes, any address that translates to Non-secure physical address space is treated as execute-never. That is, an attempt to execute an instruction from a Non-secure physical address space results in a permission fault.

3.2 Controlling output PAS

When a virtual address is translated by the MMU, the output PA space is controlled by a combination of the following:

- Current Security state
- Current Exception level
- Translation tables
- System registers

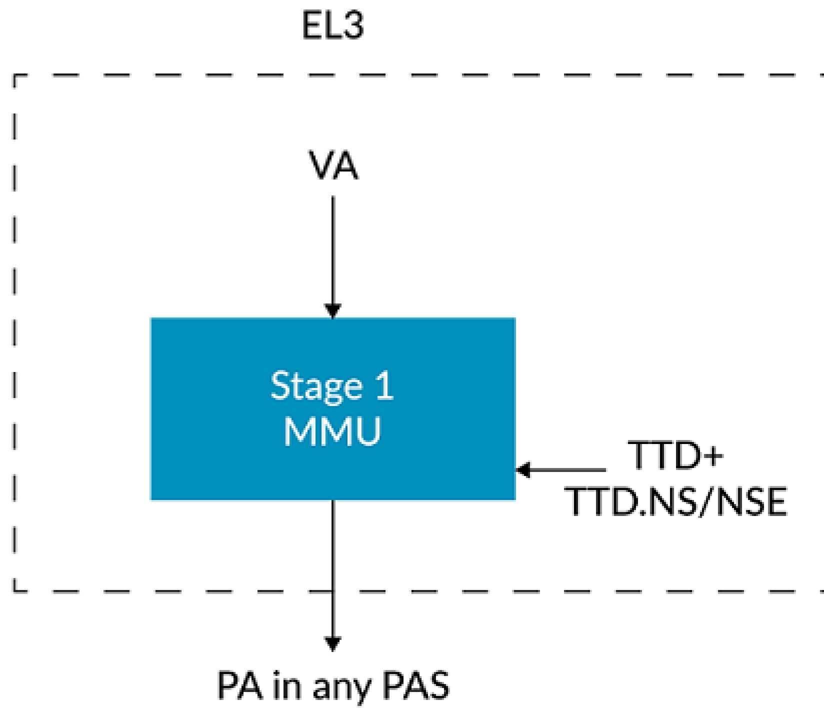
The controls that are available to software vary depending on the translation regime.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.2.1 Root state translation regimes

Root state can access all PA spaces, as shown in the following diagram:

Figure 3-2: EL3 translation regime



The EL3 stage one translation regime has two bits, NS and NSE, in the translation table entries to control the output PA space. These encodings are similar to the encodings that are used for SCR_EL3.{NSE,NS}, except that there is an encoding for Root, as shown in the following table:

TTD.{NSE.NS}	Security state
{0,0}	Secure
{0,1}	Non-Secure
{1,0}	Root
{1,1}	Realm

Note that TTD refers to Translation Table Descriptor.

RME contains the following changes to the EL3 translation regime:

- Virtual addresses can translate to physical addresses in any of the four physical address spaces
- Any address that translates to a Non-secure, Secure, or Realm physical address is treated as execute-never

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- MMU table walks can only access the Root PA space
- When the MMU is disabled at EL3, all output addresses are in the Root PA space

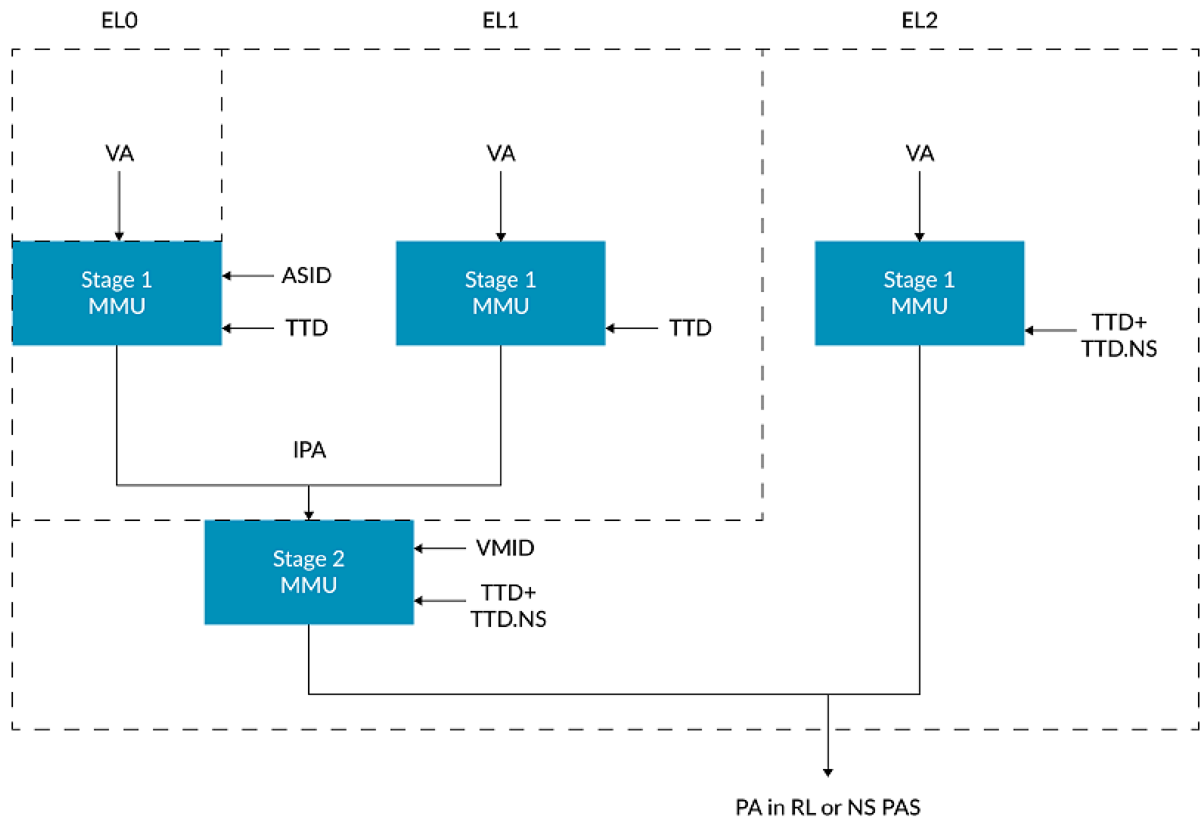
Note

EL3 is only subject to stage-1 translation regime and EL3 exists only in Root state.

3.2.2 Realm state translation regimes

Realm state can access the Realm and Non-secure PA spaces, as shown in the following diagram:

Figure 3-3: Realm translation regimes



The Realm EL0/1 translation regime has a single Realm IPA space. Therefore, there is no NS bit in the EL0/1 stage one translation table entries.

The Realm stage two TTDs include an NS bit, to map to either the Realm or Non-secure PAS. This means that, unlike Secure state, Realm state has per-page controls at stage two.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The Realm EL2 translation regime and Realm EL0/2 translation regime have stage one NS bit to control the output PA space.

The NS bit in the translation table entries was introduced by TrustZone to enable Secure state to select the output PA space. In Secure state, the NS bit is encoded as follows:

- NS=0: Secure
- NS=1: Non-secure

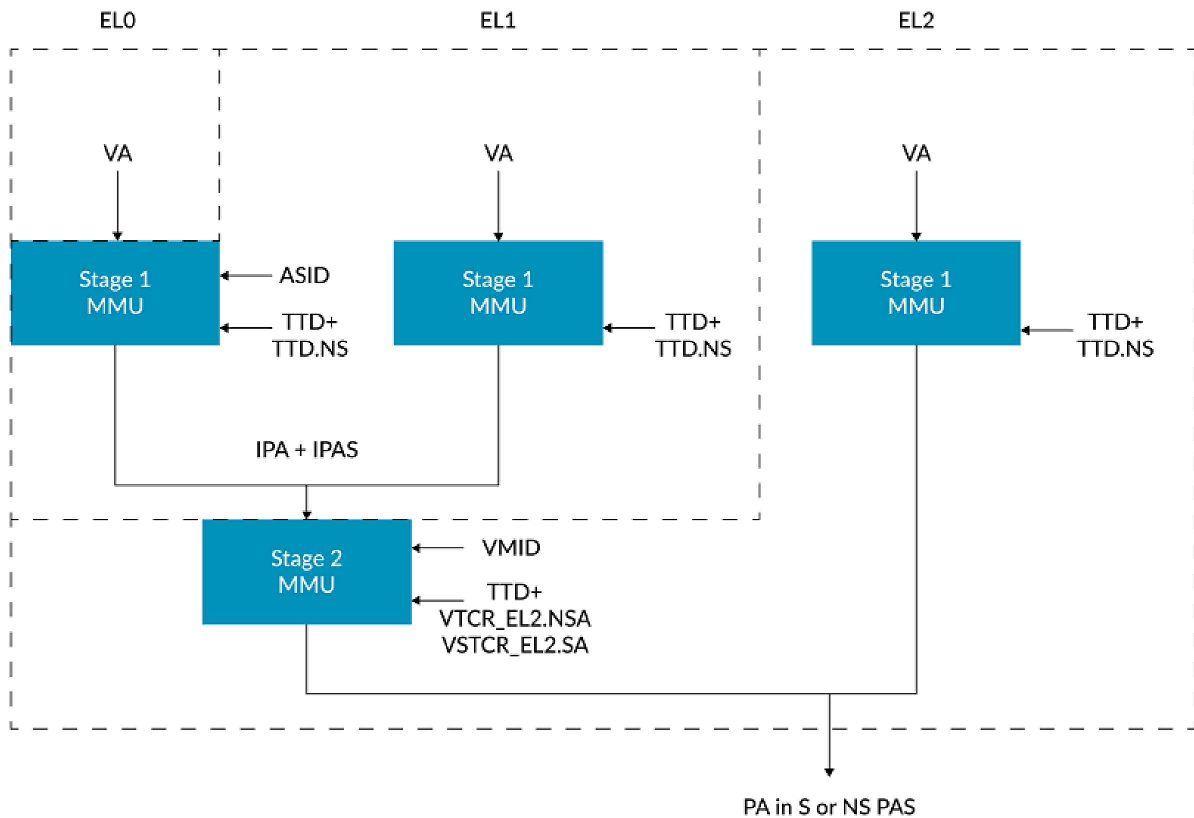
With RME, the NS field is also used in Realm EL0/1 stage two and Realm EL2/0 stage one translation tables, but is encoded as:

- NS=0: Realm
- NS=1: Non-secure

3.2.3 Secure state translation regimes

Secure state can access the Secure and Non-secure PA spaces, as shown in the following diagram:

Figure 3-4: Secure translation regimes



Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

For the Secure translation regime, the NS bit in the stage 1 translation table entries selects between two Intermedial Physical Address (IPA) spaces.

There are controls at EL2, VTCR_EL2, and VSTCR_EL2, to map each of those IPA spaces to a PA space.

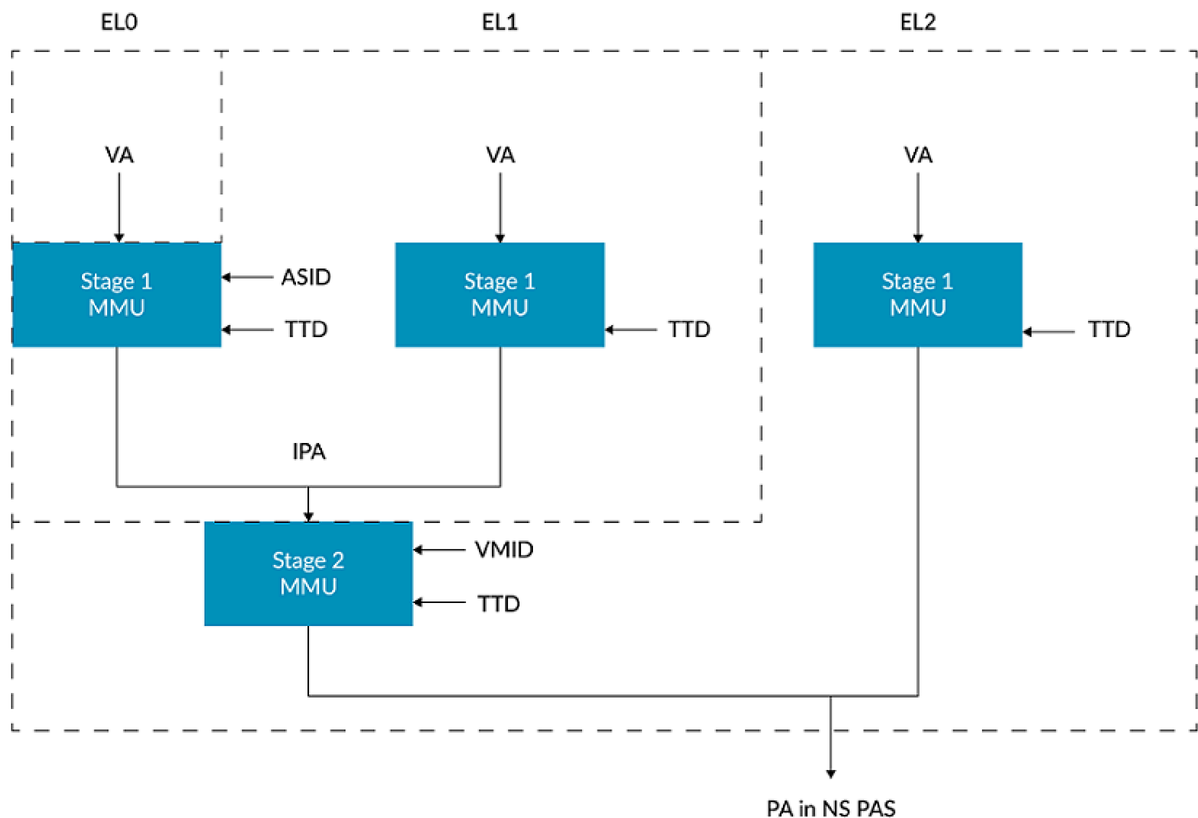
Note

In Secure state, the controls are unchanged by RME. This guide describes them for completeness.

3.2.4 Non-secure state translation regimes

The following diagram shows an overview of the Non-secure translation regimes:

Figure 3-5: Non-secure translation regimes



Non-secure state can only access the Non-secure PA space. Therefore, in Non-secure state there are no controls at stage 1 or stage 2 for controlling the output IPA space or PA space.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

3.2.5 Impact on Translation Lookaside Buffers and caches

TLBs cache recently used translations. Translation Lookaside Buffer (TLB) entries need to record which translation regime an entry belongs to. During a TLB look-up, an entry can only be returned if the translation regime in the entry matches the requested translation regime. This prevents one Security state from using the TLB entries of another Security state.

The following table shows an example simplified TLB, recording the translation regime:

Figure 3-6: Translation Lookaside Buffer

VA	Translation Regime	Security state	VMID	ASID	Descriptor
0x800000	EL3	RT	-	-	RT:0x900000
0x500000	EL2	NS	6	-	NP:0xA00000
0xC00000	EL1	S	5	3	NP:0x500000

Translation Look-aside Buffer (TLB)

Similarly, caches lines need to record the associated PA space, as shown in the following diagram:

Figure 3-7: Example cache with PAS recorded

TAG	MESI state	Data RAM
NSP: 0x800000	mEsi	0xFF0056AD, 0xCD503410, ...
SP: 0x800000	meSi	0x00000000, 0x548EDAB, ...
-	mesI	-

Example 4-way data-cache

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4. Granule Protection Checks

This section describes Granule Protection Checks (GPC) introduced by RME. GPCs enable the dynamic assignment of memory regions between different physical addresses spaces.

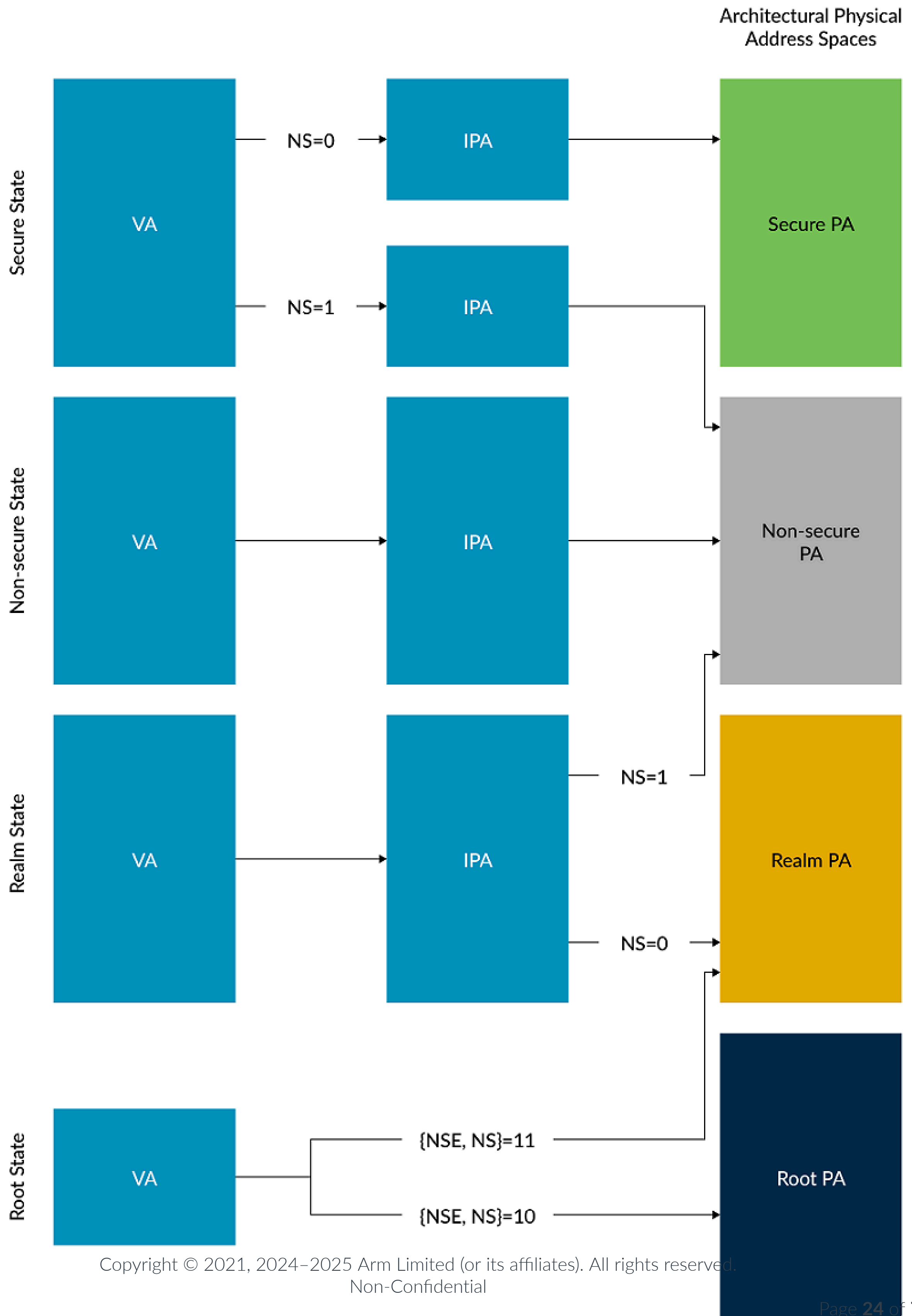
This section teaches you about the following features:

- The structure of Granule Protection Tables (GPTs)
- Fault reporting for GPCs
- Granules transition between PA spaces

As Physical addresses describes, RME provides four physical address spaces as the following diagram shows:

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-1: VA to PA mapping



Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The above diagram shows one subset of the available translation regimes.

In theory, each PA space is separate and independent and can be fully populated. In practice, most designs have a single effective PA space for DRAM regions, using the PA spaces to partition the space into regions, as shown in the following diagram:

Figure 4-2: Multiple PA spaces

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

For on-chip devices and memories, the memory system typically enforces isolation. This isolation is provided either in the end peripheral or in the interconnect. This configuration is referred to as completer-side filtering, for example:

- On-chip ROM and SRAM, which is Root-only and the interconnect enforces it. Example use cases include system boot.
- Generic Interrupt Controller (GIC). Transactions are routed to the GIC regardless of PA space. GIC internally uses the security state of the accessor to control which state and configuration is accessible.

For bulk memory, RME provides a mechanism to dynamically allocate pages to different PA spaces at runtime. For example, when starting a Realm, ownership of some memory is transferred from Non-secure state to Realm state. When that Realm is terminated the memory is reclaimed, and ownership returns to a Non-secure state.

Note

In the system architecture, the physical address space region that is assigned is called the Resource PA space.

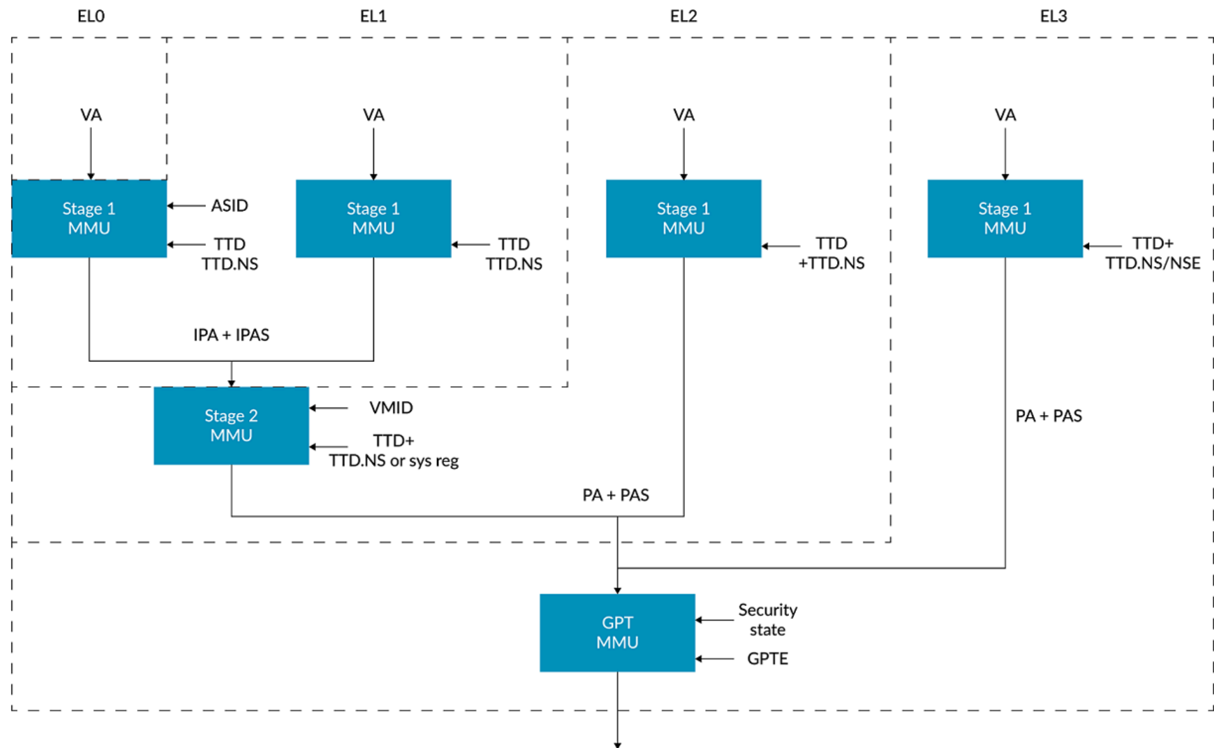
The Granule Protection Checks in the MMU enables dynamic allocation of pages to PA spaces. A set of Granule Protection Tables (GPTs) records for every location that is either of the following:

- Completer-side filtered.The MMU permits all accesses and relies on memory system checks. These checks can be carried out in either the interconnect or the peripheral.
- Allocated to a PAS:
    - The MMU only permits access where the output physical address space from VA to PA translation matches the PA space in the GPTs
    - Where the physical address space does not match, the MMU blocks the access and returns a Granule Protection Fault (GPF)

The following diagram shows the MMU after stage one and stage two translations that perform GPC

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-3: MMU translation stages



In the diagram, stages are shown as serial.However, the process is more complicated. The following table shows an example of an LDR instruction that is executed in NS_EL1. For simplicity, we assume a single table level at stages 1 and 2:

Step	Action by PE	Notes
1	LDR instruction issued	-
2	Read TTBR<n>_EL1 to get IPA of stage 1 table	Before the stage 1 descriptor can be fetched, the IPA must be translated into a PA
3	Read VTTBR_EL2 to get PA of the stage 2 table	Before the stage 2 descriptor can be fetched, the PA must have a Granule Protection Check
4	Perform Granule Protection Check on PA of stage 2 table descriptor	-
5	Read stage 2 table descriptor and translate IPA of stage 1 descriptor to a PA	We now have the PA of the stage 1 descriptor. Before it can be fetched, the PA must have a Granule Protection Check.
6	Perform Granule Protection Check on physical address of stage 1 table entry	-
7	Read stage 1 table entry and translate input VA to IPA	-
8	Perform Granule Protection Check on physical address of stage 2 table entry	Before the stage 2 descriptor can be fetched, it must have a Granule Protection Check

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

For a running system, most accesses reuse cached translations in the TLBs. However, the example highlights the interaction between the different stages of translation and Granule Protection Checks.

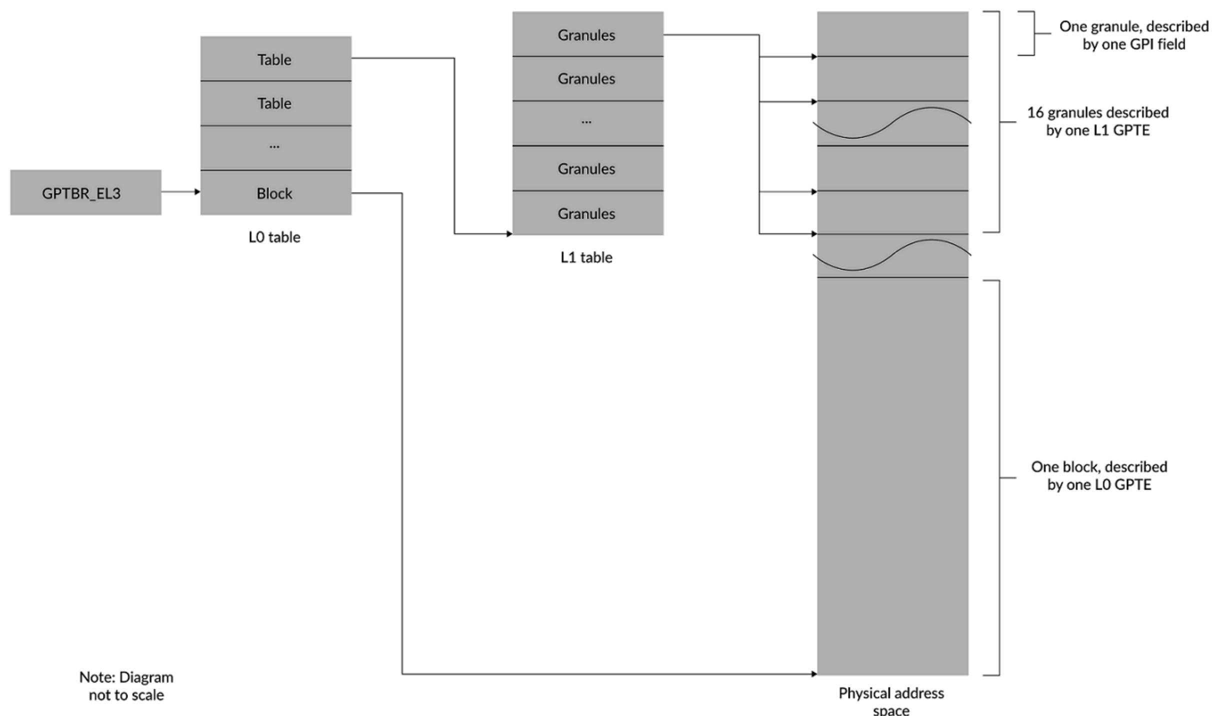
4.1 GPTs

A set of GPT tables, configures which PA space each granule is associated with. When the processor performs an access, the MMU walks the GPTs to determine whether the access is permitted. GPCs are configured using the following System registers:

- GPCCR_EL3 to configure: * Granule Protection Checks Enable * Granule size: 4K, 16K, or 64K * Size of protected region
- GPTBR_EL3 to configure the PA of the GPT, which is in the Root PA space

The following diagram shows GPTs with two-level table structure as an example:

Figure 4-4: Example simple GPT structure



GPTBR_EL3 points to the base of the level 0 table. Each level 0 table entry covers a 1GB region (note that this can be configured using GPCCR_EL3.L0GPTSZ), and can be one of the following formats:

- Block Descriptor. The block is assigned to a specific PAS or configured to enable all PA spaces.
- Table Descriptor:

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

    - The level 0 table entry is subdivided into granules representing the region, with a level 1 GPT describing each granule
    - The descriptor gives the PA of the level 1 table. The table must be in the Root PA space.

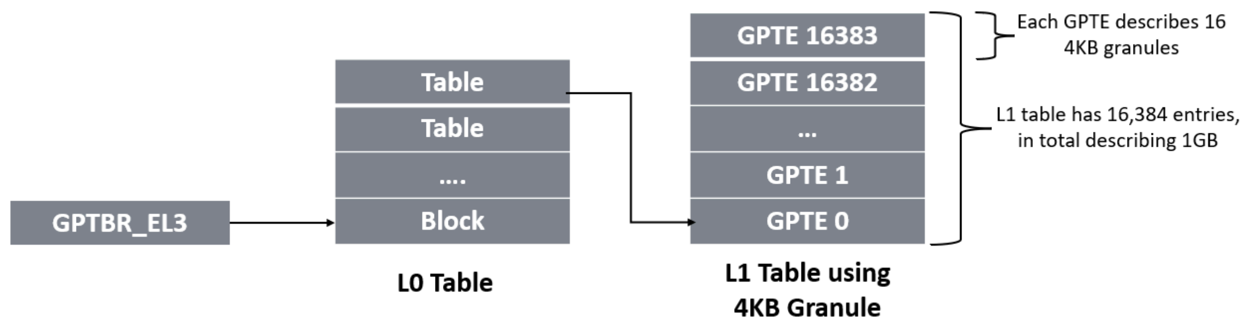
Each entry in a level 1 GPT is one of the following:

- Granule Descriptor
    - Contains 16 Granule Protection Information (GPI) fields; with each GPI field describing one granule of PA space
    - Each granule can be independently assigned to a single PA space or configured to enable all PA spaces. The latter configuration delegates responsibility to check the legality of the access to a completer-side filter.
- Contiguous Descriptor is like a Granule Descriptor, but describes larger regions. Using larger regions can enable more efficient caching in the TLBs.

The granule size is configurable through GPCCR_EL3 and matches the granules sizes that are available for the translation tables.

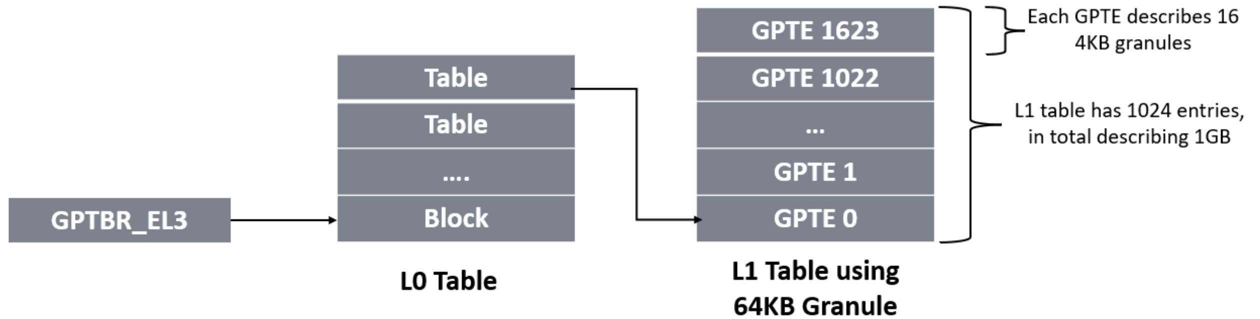
The following diagram shows the level 0 table size as 1GB,configured using GPCCR_EL3.L0GPTSZ bit fields, and level 1 table sizes using 4KB and 64KB granule size.

Figure 4-5: Effect of 4KB granule size



Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-6: Effect of 64KB granule size

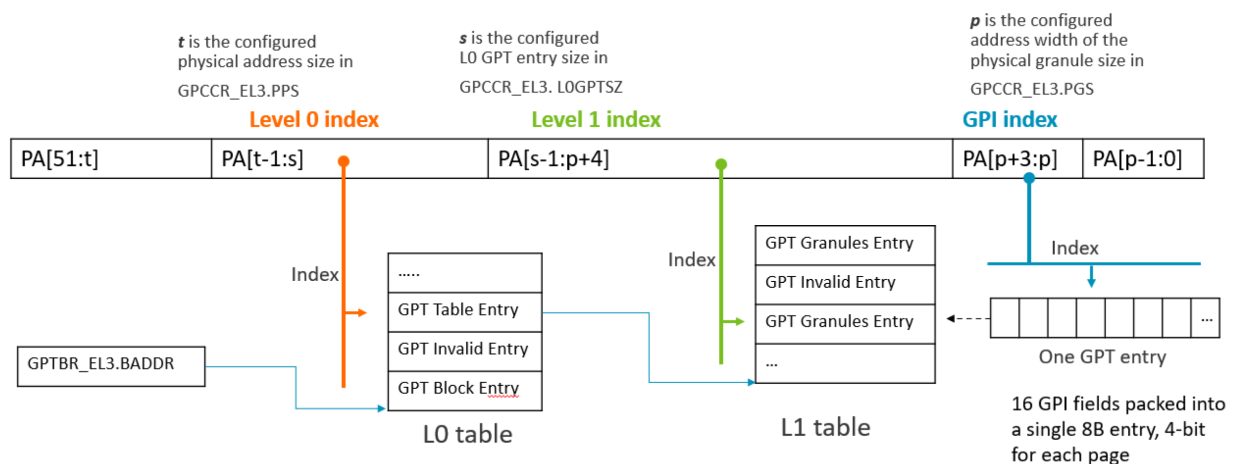


A GPTE refers to an entry in either a level 0 or level 1 GPT. GPI refers to the fields in a GPTE that describe the assigned PA space for a region of memory.

GPI entries for level 0 entries are only used in block descriptors.

For the general GPT table lookup process, see section D9.4.1 of Armv8-A Architecture Reference Manual. Here is a pictorial representation of how a GPT table lookup process happens in a system.

Figure 4-7: GPT Table Lookup



In Armv9.1 RME system architecture specifications, the L0GPT value imposed a scaling challenge of requiring >=2MB of memory space on a >= 48bit PA. To overcome this challenge, in FEAT_RME_GPC3 extension, an additional bit called Protected Physical address Size (PPS) is introduced in GPCCR_EL3 register. The GPTBR_EL3.PPS defines the region that the GPT

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

covers starting at address 0 and extends to an upper bound. Any address beyond the range that GPTBR_EL3.PPS defines is treated as belonging to the Non-secure PA space. The following table describes how the level 0 table is indexed by PA bits.

GPCCR_EL3.PPS[3:0]	Level 0 index	Usable Address space
0b0000	PA[31:s]	4GB
0b0001	PA[35:s]	64GB
0b0010	PA[39:s]	1TB
0b0011	PA[41:s]	4TB
0b0100	PA[43:0]	16TB
0b1000	PA[45:s]	64TB
0b1001	PA[46:s]	128TB
0b0101	PA[47:s]	256TB
0b0110	PA[51:s]	4PB
0b0111	PA[55:s]	64PB

4.2 GPC faults

If an access fails its GPCs, a fault is reported. Collectively, these faults are referred to as GPC faults.

The types of GPC fault are as follows:

- Granule Protection Fault (GPF): The GPT walk completed successfully, but the access was not permitted.
- GPT Walk Fault: The GPT walk failed to complete because of an invalid GPT entry.
- GPT address size fault: The GPT walk failed because of attempted access to an address beyond the configured range.
- Synchronous External abort on GPT fetch: The GPT walk failed because a read of a GPT entry returned an External abort.

GPC faults are reported as one of the following exception types:

- Data Abort exception
- Instruction Abort exception
- GPC exception

GPC exception is a new synchronous exception type introduced by RME.

Note

GPFs on accesses to the trace and Statistical Profiling Extension (SPE) buffers are handled differently. For more information, see Self-hosted trace and SPE.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.2.1 Granule Protection Faults (GPFs)

A GPF is generated when the PA space of the output address resulting from VMSA VA to PA translation does not match the PA space that the granule is assigned to in the GPTs.

For example, software attempts to access RLP:`0x8000`, but PA `0x8000` is allocated to the Non-secure PA space.

GPFs can be reported as GPC exceptions, Instruction Abort exceptions, or Data Abort exceptions as summarized in the following table:

Fault generated at	SCR_EL3.GPF	HCR_EL2.GPF	HCR_EL2.TGE	Resulting exception
EL0	0	0	0	Instruction or Data Abort, taken to Exception level 1
	0	0	1	Instruction or Data Abort, taken to Exception level 2
	0	1	x	Instruction or Data Abort taken to Exception level 2
	1	x	x	GPC exception, taken to Exception level 3
EL1	0	0	x	Instruction or Data Abort, taken to Exception level 1
	0	1	x	Instruction or Data Abort, taken to Exception level 2
	1	x	x	GPC exception, taken to Exception level 3
EL2	0	x	x	Instruction or Data Abort taken to Exception level 2
	1	x	x	GPC exception, taken to Exception level 3
EL3	x	x	x	Instruction or Data Abort, taken to Exception level 3

The exception syndrome that is provided for Instruction Aborts and Data Aborts has been extended to give information on GPFs.

A GPT walk can fail to complete, and one of the following GPC faults is reported:

- GPT address size fault
- GPT walk fault
- Synchronous External abort on GPTE fetch

Arm expects these faults to be rare in system during normal operations. These faults typically represent an Exception level 3 software error or a loss of consistency, which is likely to be fatal.

These fault types are always reported as GPC exceptions, and taken to Exception level 3.

4.3 Transitioning a granule between physical address spaces

A granule can be moved between PA spaces by updating the GPTs. The architecture specification includes the required sequences that software must follow to transition a granule between PA spaces. For more details, refer Realm Management Extension example software flows guide.

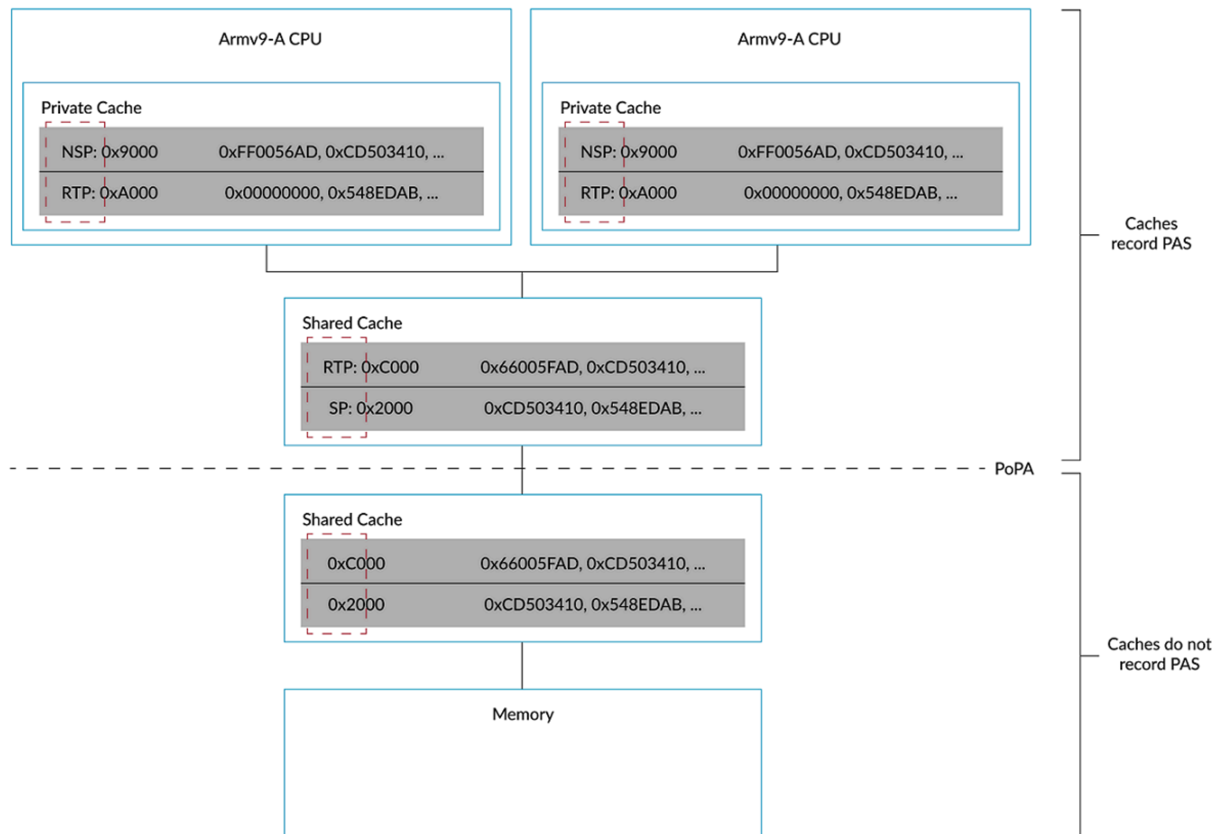
Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.3.1 Impact on caches

As part of transitioning a block or granule, Exception level 3 software ensures that copies of the location held in caches with the old PA space are removed.

To permit removal of these cache lines, RME introduces Point of Physical Aliasing (PoPA), a new conceptual point in the cache hierarchy. The PoPA is the point beyond which an access using any PA space uses the same copy in a cache or memory. The following diagram shows an example of PoPA:

Figure 4-8: PoPA example



As part of the transition flow, software cleans and invalidates the caches to PoPA. This ensures that no copies of the granule are held in caches with the old PA space.

Note

Not all systems include caches beyond the PoPA.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

4.3.2 Impact on TLBs

TLBs hold copies of recently used translations and can include the result of Granule Protection Checks. When a granule is moved between PA spaces, software must issue invalidate operations to remove any cached copies of the GPTE information. RME introduces new TLBI instructions for this purpose.

The Arm architecture does not specify how TLBs are structured, and the structure can vary between implementations. Possible approaches include one of both of the following:

- Caching the result of different stages separately
- Caching the result of multiple stages as a single entry

Software does not need to be aware of which approach is implemented.

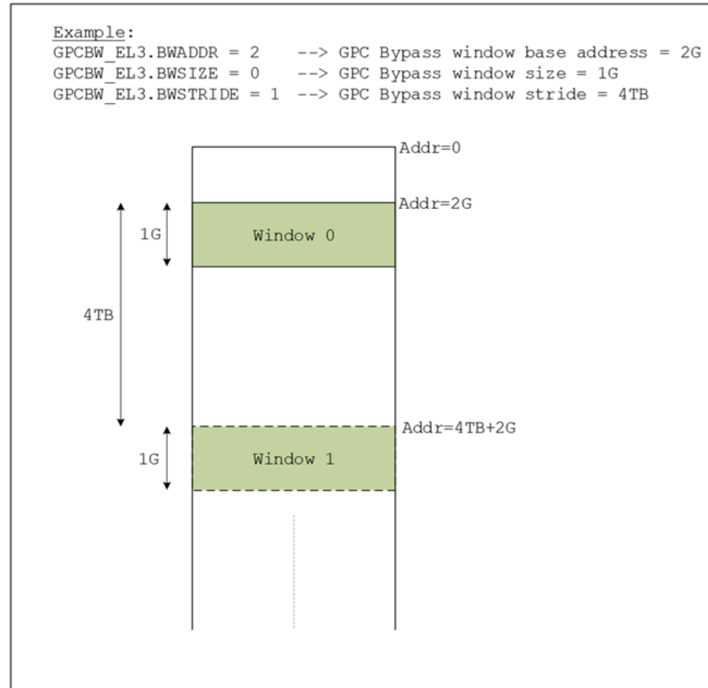
4.4 GPC bypass windows

With FEAT_RME_GPC3 extension, architecture enables programming windows in memory space for which the effective GPI value is “All Access Permitted” or in other words, GPC checks are bypassed and there is not need to perform GPT access. This type of windows in memory space can be used in completer-side PAS filter (such as Boot ROM, SRAM and so on).

The GPC bypass window can be duplicated in memory map across a programmable stride,for example: 64TB. Typically in a multi-socket topology, each socket has this window placed at same relative offset. Window parameters are configured using GPCBW_EL3 register as shown in following figure:

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 4-9: GPC Bypass windows



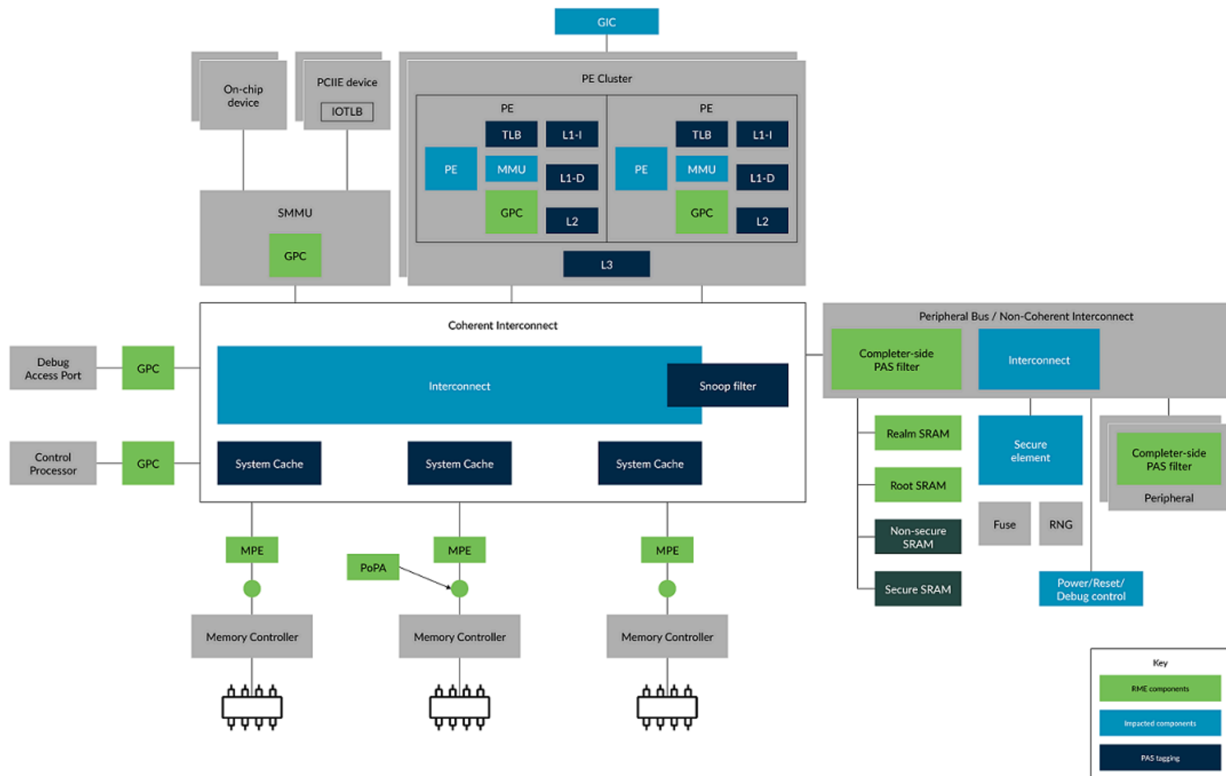
Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5. System architecture

RME is more than just a set of processor features. To take advantage of the features introduced in RME, additional support is required in other system components.

The following diagram shows an example system and the components affected by the introduction of RME.

Figure 5-1: Example system with impacts of RME



5.1 Main memory protection

RME-enabled systems include memory encryption and, potentially, memory integrity checking. The baseline requirement is encryption for external memory, using a separate encryption key or tweak for each PA space. This encryption also provides spatial isolation using an address tweak.

This guide uses the term Memory Protection Engine (MPE) to describe the component that provides external memory encryption and integrity services.

For more information about RME System Architecture, see - SMMU architecture, - Memory Encryption Contexts extension - SMMU implementation:

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.2 MPAM

Prior to RME, the Memory Partitioning And Monitoring extension (MPAM) defined independent PARTID spaces for Non-secure and Secure states distinguished by the MPAM_NS value. MPAM_NS selects between Non-secure and Secure banks of resource usage monitors. Each monitor can be sensitive to a PARTID or PARTID and PMG, or PMG alone. The PARTID space as conveyed by MPAM_NS is used to choose which monitor bank tracks the resource usage.

Sharing the PARTID space between Non-secure and Realm Security states can be required for host resource management considerations. In some cases, sharing PARTID space can leak information on workload behavior. For example, monitoring Realm accesses using a Non-secure monitor is a potential side channel. Allocating cache partitions to a Realm is a mechanism that could be used in cache timing attacks.

With RME, MPAM_NS is replaced with a 2-bit MPAM_SP. This MPAM space field enables systems to implement four independent PARTID spaces, one for each Security state. MPAM_SP also defines how multiple Security states can share a single PARTID space. Security states enable the system integrator to decide how MPAM is implemented in an Arm CCA system. This implementation decision is based on the risks exposed by each MPAM Memory System Component (MSC).

5.3 RAS

A key requirement of RAS support for RME-enabled systems is to maintain the security isolation boundaries, of confidentiality and integrity, that RME provides. A challenge with this requirement is that you might want RAS to be triaged in the Non-secure state, in hypervisor or kernel code. This implies that PEs running in Non-secure state have direct access to sanitized Error Record registers.

RME introduces the idea of confidential information, which is information that is not accessible to the current Security state under normal operation. For example, Non-secure state does not normally have access to contents of a Secure or Realm memory location.

RME places restrictions on what information is exposed by RAS:

- Root state can see information belonging to any Security state
- Secure state can see confidential information belonging to Secure or Non-secure state, but not Root or Realm
- Realm state can see confidential information belonging to Realm or Non-secure state, but not Root or Secure
- Non-secure state cannot see confidential information belonging to any other state

Not all information that is associated with an error is considered confidential. Usually, the address of the resource and the severity can be reported. For example, an RAS fault on a Realm location can be reported to Non-secure state. The fault record can report the error type, and the address of the location that suffered the fault. It will not, however, be allowed to expose anything about the contents of that memory location.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

5.4 Granule Data Isolation (GDI)

The FEAT_RME_GDI extensions introduce the support for Granular Data Isolation (GDI). GDI adds two new PA spaces, that a memory location can be assigned to (from a RME System Architecture perspective):

- Non-Secure Protected (NSP)
- System Agent (SA)

To describe these new PA spaces, an NSE2 bit is introduced, alongside the NSE bit which was introduced by RME. The following table shows the summary of PAS:

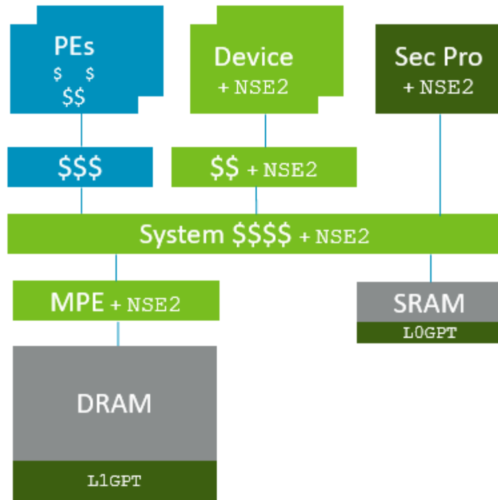
NSE2	NSE	NS	PA space	GPT name
0	0	0	Secure	S
0	0	1	Non-secure	NS
0	1	0	Root	RT
0	1	1	Realm	RL
1	0	0	System Agent	SA
1	0	1	Non-secure Protected	NSP
1	1	0	Reserved - No access	NA6
1	1	1	Reserved - No access	NA7

Note that the processor is not permitted to perform memory access to NSP or SA PA spaces from any security state. There is no way to configure the Stage 1 and Stage 2 translation tables on the processor to output an address in either SA or NSP PA spaces. Only a limited system-specific set of trusted devices can generate accesses into those physical address spaces.

GDI enables software to manage buffers allocated to other devices, while maintaining the confidentiality of those buffers. For example, the NSP PAS could be used by trusted accelerators to process data while guaranteeing the data is not accessible to software. An example use case is a trusted-media pipe, where it is important to keep the decrypted media confidential. The following figure shows the expanded system to support NSE2:

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Figure 5-2: Expanded system to support NSE2



With the introduction of NSE2 bit, the system interconnect also needs to be expanded to carry the value of NSE2 bit for memory addressing and for some of the Cache maintenance operations (such as DC CIPAPA and DC CIGDPAPA).

5.5 Explicit PAS control for CXL-TSP

In the context of CXL consortium and CXL 3.1 specification, the "TE state bit" refers to a bit that indicates the trust level for CXL cache devices. In a RME enabled system, due to the complexity of the CXL memory topology, there is a need for setting the "CXL TE-state" bit from processor for completer-side GPC.

The FEAT_RME_GPC3 extension adds new instruction called `APAS Xt`. This instruction enable the CXL-TE-State check to be set for a given memory location and will also associate a memory location with a PA space.

System Instruction Mnemonic	Instruction	Notes
APAS Xt	Associate PA space	EL3 Only

The granularity of APAS instruction applies to an implementation defined memory address range which should be in the alignment of 64 bytes.

Note

If APAS is issued to a location that does not support the APAS instruction, or to a location that cannot be associated with the indicated PA space, then the instruction has no effect on the location and does not generate an External abort.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

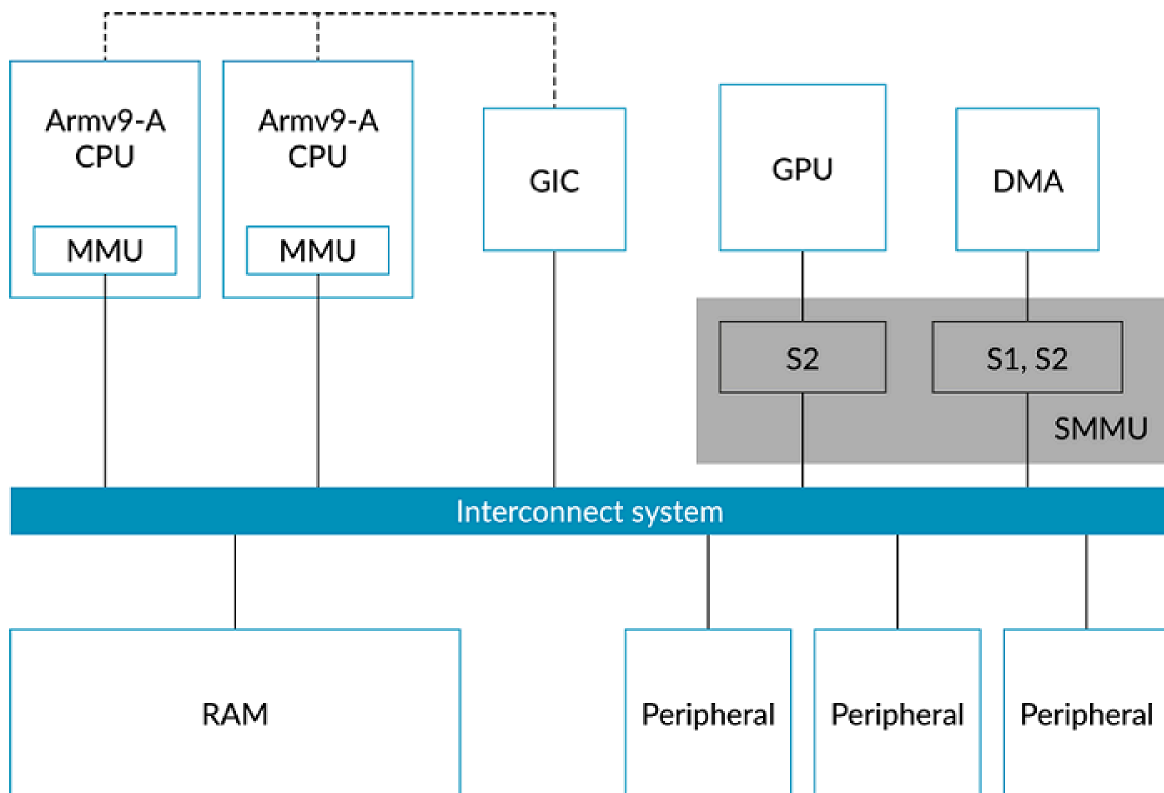
6. SMMU architecture

The SMMU architecture is extended to support RME and RME Device Assignment (DA). In this section of the guide, we describe how SMMUs are used in an RME-enabled system and the key changes to the SMMU architecture.

6.1 SMMUs in an RME-enabled system

A system might include several devices that can independently access memory, such as DMA controllers or GPUs. The following diagram shows a simplified system:

Figure 6-1: Example system before RME



Any device that can access memory, and is therefore a Requester, must be subject to the physical address space isolation guarantees of TrustZone and RME.

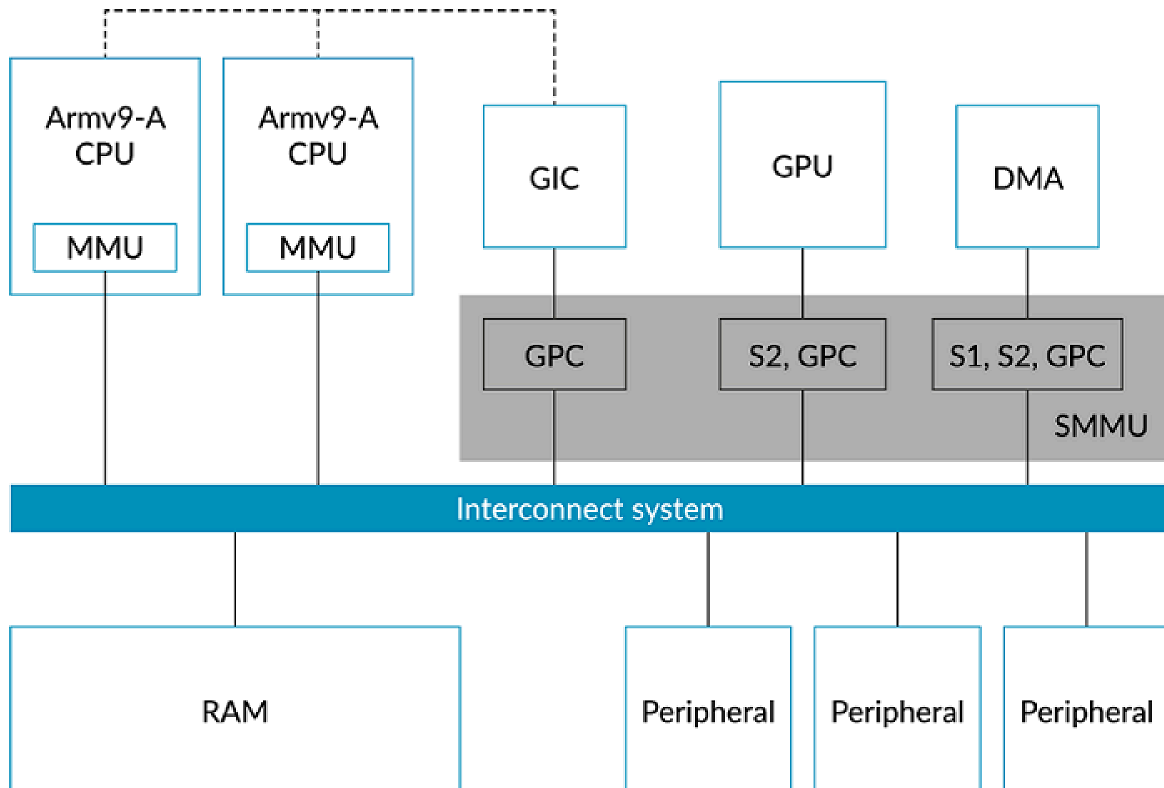
For regions that rely on completer-side filtering, this isolation guarantee is achieved in the memory system. The Requester specifies the PA and PAS space that it wants to access. The memory system or target peripheral determines whether the access is permitted. This process is unchanged from TrustZone.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

RME provides support to dynamically assign pages of memory to different PASs. Access by all devices to assignable locations must be checked against the GPT. For the CPU, this access is handled by the MMU. For other devices, access is handled by the SMMU.

The following diagram shows an example system using the SMMU to provide GPCs:

Figure 6-2: Example system with SMMU



In the diagram showing the simplified example system before RME, the GPU and DMA use SMMUs to provide translation. In an RME enabled system, the SMMU also provides GPCs. However, the GIC is an example of a device that has not previously been connected through an SMMU. The GIC can access memory and needs GPCs. In the RME enabled example system, the GIC is connected through the SMMU, but the SMMU only provides GPCs.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.2 Changes to the SMMU architecture for RME

Key features of SMMU for RME include:

Client devices and SEC_SID

In the SMMU architecture, SEC_SID identifies the Security state of a client device. Only Secure and Non-secure client devices can be identified. Root or Realm client devices are not supported, so device accesses to Realm memory are blocked by the SMMU.

Clients devices that do not have a StreamID

In the SMMU architecture, a StreamID is used to identify the client device that sent a transaction, and to determine what translations to perform. RME extends the SMMU architecture to support transactions without a StreamID, called NoStreamID accesses or devices. These transactions are subject to Granule Protection Checks, but not stage 1 or stage 2 translation. Arm expects this support to be used for devices like the GIC in the example system in SMMUs in an RME-enabled system. That is, this support is used for devices that are traditionally not connected to an SMMU but whose accesses need GPCs.

SMMU-originated accesses

Accesses originating from the SMMU, for example reads of the Stream Table, are subject to GPCs. These checks are the same as accesses as part of a stage 1 or stage 2 walk by the PE MMU, which are subject to GPCs. Where an SMMU-originated access triggers a fault from the Granule Protection Checks (GPC), it is reported as though the SMMU experienced an External abort.

6.3 Changes to the SMMU architecture for RME DA

The SMMU for RME DA introduces features that enable the association between devices and software executing in the Realm Security state. It supports the Realm programming interface, ensuring that devices can be associated with realms securely.

Key features of SMMU for RME device assignment include:

Support for the Realm programming interface

Enables SMMU streams associated with Realm-assigned devices to be configured securely by the Realm Management Monitor (RMM). This is basically a clone of the Non-secure programming interface but with Realm translation regime behavior at stage 1 and stage 2. For RMM simplicity, the architecture requires or prohibits some subset of features.

Memory Encryption Contexts (MEC)

Supports the encryption of memory contexts associated with the Realm, providing an additional layer of security.

Device Permission Table (DPT)

Records the assignment of a physical granule of memory to a specific Realm (the latter indicated by a VMID value). This ensures that a Realm-assigned device which implements ATS and issues translated requests can only access the memory belonging to its assigned Realm.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

These features collectively ensure that devices assigned to a Realm can operate securely and efficiently, adhering to the strict security protocols necessary for Realm operations.

6.4 SMMU support for GPC

Access to physical memory is governed by GPC, which ensure that only authorized transactions can reach specific memory regions. This mechanism supports the dynamic assignment of pages to different PAS, offering greater flexibility in memory management. By eliminating the need for completer-side filtering based on fixed memory carveouts, the system achieves more efficient and scalable memory access control. GPCs also enforce that the security attribute of each translated transaction is validated, guaranteeing that access to the target address complies with the defined security policies.

An SMMU that implements the RME is subject to GPC when it accesses to all physical addresses if GPC is enabled. Access by all client devices to a physical address must be checked against the GPT.

The following lists the types of SMMU memory accesses which are subject to GPC:

- SMMU-originated accesses
    - Configuration and translation table walks
    - Hardware Translation Table Updates (HTTU)
    - Device Permission Table (DPT) Walks
    - Queue accesses (Event/Command/PRI)
    - Message-Signalled Interrupts (MSIs)
- Client-originated accesses
    - GPC on the final translated address

This behavior of the SMMU is the counterpart of FEAT_RME in the A-profile CPU architecture. For more details about GPC, see Granule Protection Checks.

Fetching Granule Protection Table (GPT) information is not subject to GPC.

6.4.1 How GPC is controlled in an SMMU

To support GPC, an SMMU with RME adds a Root Control Page which is accessible only in the Root PAS. Its base address is distinct from addresses of registers accessible in other PASs.

The `SMMU_ROOT_CR0` register within the Root Control Page controls GPC settings. The `SMMU_ROOT_CR0.GPCEN` field determines whether GPC is enforced:

- When the `GPCEN` bit is set to `0b0`, all accesses bypass GPC.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- when the GPCEN bit is set to 0b1, all client and SMMU-originated accesses, except for GPT walks, are subject to GPC.

Note

The SMMU_ROOT_CR0.ACCESSEN bit controls whether to enable accesses initiated by the SMMU and client devices. It takes precedence over SMMU_ROOT_CR0.GPCEN. Therefore, software typically enables the SMMU_ROOT_CR0.ACCESSEN bit as well.

SMMU_ROOT_GPT_BASE serves as the control register for the GPT base address. SMMU_ROOT_GPT_BASE is the counterpart of GPTBR_EL3 in the A-profile CPU architecture.

6.4.2 Invalidation of GPT information

TLBs cache recently used address translations and might also store the results of GPC. When a granule is reassigned between physical address spaces, software must issue invalidation operations to clear any stale GPT entry data from the TLBs to maintain consistency and security. An SMMU with RME participates in broadcast TLBI *PA* instructions from PEs that are executing in EL3 regardless of the values of SMMU_IDR0.BTM and SMMU_(S_)CR2.PTM.

6.4.3 GPC faults

These are classes of GPC fault:

- Granule Protection Fault (GPF)
  - Attempted access for a Location using the wrong PAS
  - Attempted translation table walk fetching from a Location with the wrong PAS
  - Attempted access with PA size exceeding the configured size for the GPT
- GPT lookup error
  - Mis-programming SMMU_ROOT_GPT_BASE or SMMU_ROOT_GPT_BASE_CFG
  - A GPT entry with invalid values
  - A GPT fetch that encounters an external abort
- Faults arising from RAS errors
  - A GPT fetch that fails for a RAS-related reason

A GPF is reported in SMMU_ROOT_GPF_FAR, while a GPT lookup error is reported in SMMU_ROOT_GPT_CFG_FAR. A fault arising from a RAS error while fetching a GPT Entry is reported in the same manner as if the SMMU experienced an External abort while fetching a GPT Entry.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.4.4 GPC interrupts

An SMMU with RME has two additional edge-triggered wired interrupts. The following table shows these two interrupts and their trigger conditions:

Interrupt source	Trigger condition
GPF_FAR	An error becomes active in `SMMU_ROOT_GPF_FAR`.
GPT_CFG_FAR	An error becomes active in `SMMU_ROOT_GPT_CFG_FAR`.

6.4.5 GPC for NoStreamID devices

NoStreamID devices only access physical address space and are not associated with any stage 1 or stage 2 translation configuration. This means they operate directly on physical addresses without the typical translation stages that other devices might go through. Accesses from NoStreamID devices are subject to GPC. This ensures that even though these devices do not undergo typical address translations, their access to memory is still controlled and checked against the granule protection settings. The GPC fault reporting behaviour for accesses from NoStreamID devices is the same as for regular client-originated accesses.

6.5 Support for Realm programming interface

Support for the Realm programming interface has been introduced in the SMMU for RME DA. Similar to the Non-secure and Secure programming interfaces, the Realm programming interface has the following features:

- A Realm StreamID namespace
- Support for the Realm translation regimes
- Realm register pages
- Queues and tables for Realm state
    - Realm Command queue
    - Realm Event queue
    - Realm PRI queue
    - Realm Stream table
    - Realm CD table

6.5.1 Translation process overview

An incoming Realm transaction follows these steps:

1. If accesses from the SMMU and client devices is not enabled (`SMMU_ROOT_CR0.ACCESSEN = 0b0`), the transaction is processed in a way that is consistent with any access to a physical

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

address experiencing a GPF. If accesses from the SMMU and client devices is enabled (`SMMU_ROOT_CR0.ACCESSEN = 0b1`), the transaction is not terminated by this mechanism.

2. If the translation is globally disabled (`SMMU_R_CR0.SMMUEN == 0`), the transaction is terminated with an abort.
3. If the global bypass does not apply, the configuration is determined:
    a. A Stream Table Entry (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 Context Descriptor (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.
4. Translations are performed if the configuration is valid.
    a. If stage 1 is configured to translate, the CD contains a stage 1 translation table base pointing to the base address of a table which is walked. This might require stage 2 translations, if stage 2 is enabled for the STE. If stage 1 is configured to bypass, the input address is provided directly to stage 2.
    b. If stage 2 is configured to translate, the STE contains a stage 2 translation table base which is used to perform the stage 2 translation. Stage 2 translates the output of stage 1, if stage 1 is enabled, or translates the input address if stage 1 is bypassed. If stage 2 is configured to bypass, the stage 2 input address is provided as the output address.
5. When a transaction passes all translation stages then the translated address with the relevant memory attributes is forwarded into the system.

Accesses to all physical addresses, except for GPT walks, are subject to GPC if GPC is enabled.

6.5.2 Stream security

A stream can be Secure, Non-secure, or Realm, which is determined by the input signal `SEC_SID`. The following table shows how the input signal `SEC_SID` determines the Security state of a stream.

Table 6-2: Stream Security determination

SEC_SID value	Description
`0b00`	The stream is a Non-secure stream and uses the Non-secure registers and the Non-secure Stream Table
`0b01`	The stream is a Secure stream and uses the Secure registers and the Secure Stream Table
`0b10`	The stream is a Realm stream and uses the Realm registers and the Realm Stream Table

In a Confidential Compute Architecture (CCA) system with SMMU extension for RME DA, a device interface might operate in a trusted or untrusted mode:

- When operating in a untrusted mode, `SEC_SID` = Non-secure
- When operating in a trusted mode, `SEC_SID` = Realm

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.5.3 StreamWorld changes

An SMMU with RME does not support the EL3 StreamWorld. An SMMU with RME DA introduces these new StreamWorlds:

Realm-EL1

Properties equivalent to Realm EL1&0 translation regime in the A-profile CPU architecture.

Realm-EL2

Properties equivalent to Realm EL2 translation regime in the A-profile CPU architecture. This is when E2H is not used, and translations do not have an ASID tag.

Realm-EL2-E2H

Properties equivalent to Realm EL2&0 translation regime in the A-profile CPU architecture. This is when E2H is used, and translations have an ASID tag.

DPT use is supported for StreamIDs configured to use StreamWorld Realm-EL1, but not supported for those using StreamWorld Realm-EL2 or Realm-EL2-E2H. For more details about DPT, see DPT.

6.5.4 Output physical address space for a Realm transaction

Consistent with the Realm translation regimes in the A-profile CPU architecture, the output physical address space of a transaction on a Realm stream is as follows:

Configuration	Output address space determination
SMMUEN == 0	Transaction aborted
Stream bypass	STE.NSCFG
EL1 stage 1 only	always Realm PAS
EL1 stage 1 and 2	S2_TTD.NS
EL1 stage 2 only	S2_TTD.NS
EL2 or EL2-E2H stage 1	S1_TTD.NS

TTD means Translation Table Descriptor.

The Realm EL1&0 translation regime has a single intermediate physical address space, the Realm intermediate physical address space. Therefore, there is no NS bit in the EL0/1 stage 1 translation table entries. The Realm stage 2 TTDs include an NS bit, to map to either the Realm or Non-secure physical address space.

The Realm EL2 translation regime, and Realm EL2&0 translation regime, have stage 1 NS bits to control the output physical address space.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.5.5 Data structures changes

The SMMU for RME DA introduces some changes in data structures:

- Realm Stream Table Entry
  - A Realm STE has the same format and meaning as a Non-secure STE, except that all pointers from a Realm STE belong to the Realm PAS.
  - New fields and encodings are added to support DPT. For example `DPT_VMATCH` and `EATS`.
- Realm Context descriptor
  - Realm CD have the same format and meaning as a Non-secure CD, except that all pointers from a Realm CD belong to the Realm IPA space.
- Realm Command queue
  - All commands apply to Realm `SEC_SID` only.
  - Any command with a StreamID is interpreted as a Realm StreamID.
  - TLB invlidation commands apply to Realm entries.
  - New DPT maintenance commands (`CMD_DPTI_ALL` and `CMD_DPTI_PA`) are added.

6.5.6 Realm Register Pages

The SMMU for RME DA adds two consecutive 64KiB Realm Register Pages, Realm Register Pages 0 and Realm Register Pages 1, which are only accessible in Realm and Root PA spaces. The base address of Realm Register Page 0 can be derived from `SMMU_ROOT_IDR0.BA_REALM`. The registers introduced for the Realm state behave similarly to those for the Non-secure state, with a few minor exceptions made for RMM simplicity. These registers mainly serve the following functions:

- Reporting implemented features
- Configuring settings for structures and queues
- Providing top-level control, such as enabling the SMMU and queues
- Configuring interrupts
- Reporting global errors
- Performing address translation operations

6.5.7 Interrupts

The SMMU for RME DA introduces new interrupts, which can be wired, MSI, or both. The following table shows the conditions under which wired interrupts are triggered for various interrupt sources:

Interrupt source	Trigger condition
Realm Event Queue	Triggers when the queue goes from empty to non-empty.
Realm PRI Queue	Triggers based on the `SMMU_R_PRIQ_IRQ_CFG2.LO` bit.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Interrupt source	Trigger condition
Realm CMD_SYNC	Triggers when a `CMD_SYNC` command completes with `CMD_SYNC.CS == 0b01 (SIG_IRQ)`.
Realm GERROR	Triggered on GERROR activation.

The following table shows how MSIs are triggered for various interrupt sources:

Interrupt source	Trigger condition
Realm Event Queue	Same as wired, but must be enabled in `SMMU_R_EVENTQ_IRQ_CFG{0,1,2}`.
Realm PRI Queue	Triggered based on `SMMU_R_PRIQ_IRQ_CFG{0,1,2}` configuration.
Realm CMD_SYNC	Triggers when a CMD_SYNC command completes and `CMD_SYNC.MSIAddress` is non-zero.
Realm GERROR	Same as wired, but must be enabled in `SMMU_R_GERROR_IRQ_CFG{0,1,2}`.

Each MSI for the Realm programming interface targets either the Realm physical address space or the Non-secure physical address space. The following table shows how the physical address space is determined:

Register/command	Bit	Meaning
`SMMU_R_x_IRQ_CFG0`	`NS`	`0b0`: Realm physical address `0b1`: Non-secure physical address
`CMD_SYNC`	`MSI_NS`	`0b0`: Realm physical address `0b1`: Non-secure physical address

6.6 DPT

When Full ATS is enabled for a StreamID, the SMMU acts as a Translation Agent for the PCI device interface associated with that StreamID. It returns physical addresses to the device as part of ATS Translation Completions.

If the device issues an ATS Translated transaction to a physical address, the SMMU architecture did not previously provide a mechanism to ensure that such a transaction only accesses a region of memory that the device interface has permission to access.

The DPT provides a mechanism to enforce the association between granules of physical address space and the memory footprint of virtual machines. The DPT is independently optional for each of the Non-secure and Realm programming interfaces. This guide focuses on the Realm programming interface. The DPT enforces the association between physical granules and a Realm so that it prevents ATS-translated transactions accessing the memory of Realms not associated with the device.

6.6.1 Split-stage ATS versus Full ATS with DPT

In Split-stage ATS, the SMMU only returns a stage 1 translation to the ATS device. This enables the ATS device to cache stage 1 translations in its Address Translation Cache (ATC) and provides

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

it with unchecked access to the intermediate physical address space. However, access to physical memory is still protected by stage 2 translation, which is fully managed by the SMMU and still applies to ATS-translated traffic that has only been subject to stage 1 translation. This mechanism provides some advantages of Full ATS, such as reduced translation latency for stage 1 translation, but with better security. However, it has functional limitations as it prevents devices from being fully coherent and increases complexity for peer-to-peer transactions as these must now be subject to stage 2 translation in the SMMU.

Full ATS with DPT checks is a mechanism that combines the security advantages of Split-stage ATS with the flexibility of Full ATS. When this mechanism is selected, ATS-translated transactions are subject to DPT checks but no longer need to undergo a whole stage of translation, as is necessary when using Split-stage ATS. DPT checks ensure that the StreamID of the translated transaction is permitted to access the target address. This overcomes the limitation of Full ATS, even when GPC is enabled, where one Realm can still access the physical memory of another Realm. The DPT is a lookup table in memory that specifies the permitted VMID and access permissions for each physical memory region. Entries from this table can be cached by the SMMU to avoid the latency of performing DPT walks.

Feature	Split-stage ATS	Full ATS with DPT checks
Access to physical memory	Protected by stage 2 translation	Controlled via DPT checks
Security level	High: stage 2 ensures isolation	High: DPT ensures access control
Translation Latency	Reduced for stage 1 only	Reduced overall due to bypassing stage 2
Device Coherency	Limited: devices are not fully coherent	Better: closer to Full ATS functionality
Peer-to-Peer Transactions	Complex: must go through SMMU for stage 2	Simplified: DPT checks allow direct access if permitted
Security vs. Performance Trade-off	Prioritizes security with some performance gain	Balances security and performance

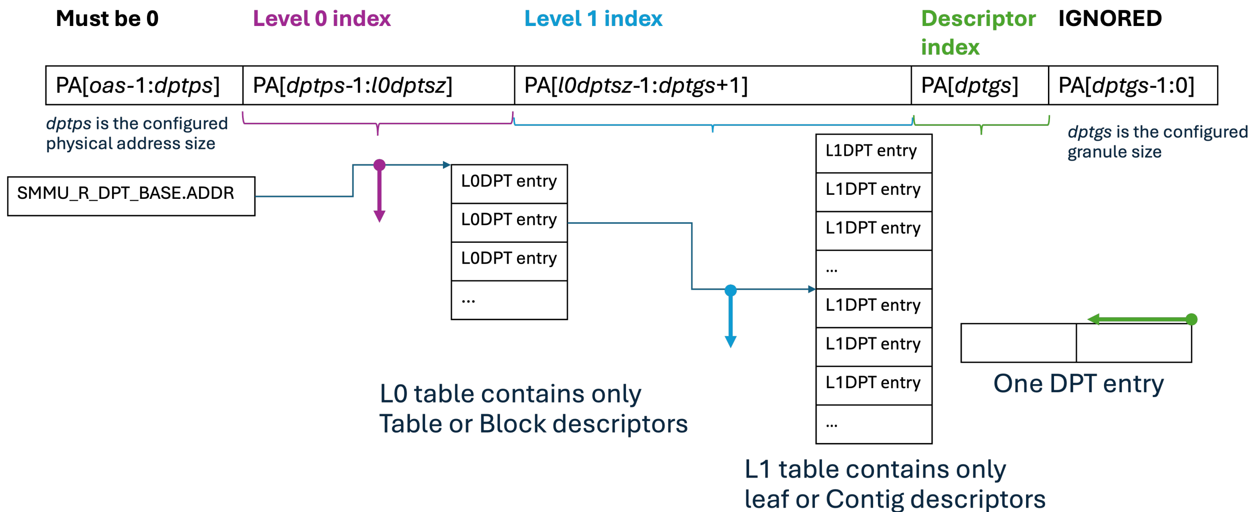
In summary, Split-stage ATS provides a higher level of security by ensuring that stage 2 translations are still applied, so protecting physical memory access. However, Full ATS with DPT checks offers a balance between security and performance. It enables ATS-translated transactions to bypass a stage of translation while still being checked against a DPT to ensure proper access permissions.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.6.2 DPT lookup

The following figure shows how to find the corresponding DPT entry through the PA from ATS-translated transactions.

Figure 6-3: DPT lookup



In the figure above, the placeholder values are:

- oas is the decoded value of `SMMU_IDR5.OAS`.
- dptps is the decoded value of `SMMU_R_DPT_BASE_CFG.DPTPS`.
- l0dptsz is the decoded value of `SMMU_R_DPT_BASE_CFG.L0DPTSZ`.
- dptgs is the decoded value of `SMMU_R_DPT_BASE_CFG.DPTGS`.

The algorithm for the DPT walk process is as follows:

1. The SMMU_R_DPT_BASE register points at the base of the level 0 table.
2. The SMMU uses bits `[dptps-1:l0dptsz]` of the input PA as the index into the level 0 table.
3. If the level 0 entry is a No access, Block or invalid entry, then the DPT walk is complete. Otherwise, the level 0 entry is a level 0 Table entry and it contains a pointer to the base of a level 1 table.
4. The SMMU uses bits `[l0dptsz-1:dptgs+1]` of the input PA as the index into the level 1 table determined in the previous step.
5. The SMMU decodes the level 1 entry according to the rules for a level 1 descriptor. Bit `[dptgs]` of the input PA might be used to select between the upper and lower half of the descriptor, depending on the value of `A[1:0]` in the descriptor.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.6.3 DPT descriptors

These are DPT descriptor formats:

- Level 0 No access entry:
    - This descriptor format indicates that no access is permitted to the granule.
    - The descriptor is marked by setting bits `[1:0]` to `0b00`
- Level 0 Block entry:
    - The descriptor includes fields for VMID and access control (AC), which determine the output PA space and access permissions.
    - Bits `[1:0]` are set to `0b01` to denote a Block descriptor.
- Level 0 Table entry:
    - The descriptor includes an address field pointing to the next level of the table.
    - Bits `[1:0]` are set to `0b11`, indicating a Table descriptor.
- Level 1 entry:
    - This descriptor format is used at level 1 and includes fields for VMID, access control (AC), and write permissions (W).
    - The descriptor also includes a field `A[1:0]` which is used to select between the upper and lower half of the descriptor. Depending on the value of `A[1:0]`, different parts of the descriptor are used to govern access to the respective granules.

The fields in DPT descriptors describes the accessibility of each granule of configured physical address space as one of the following:

- No access is permitted to the granule.
- The granule is accessible for accesses associated with some specific VMIDs.
- The granule is accessible regardless of VMID association.

6.6.4 DPT caching

The DPT cache stores the results of DPT checks, which determine the accessibility of memory regions based on the permissions associated with specific StreamIDs. This caching mechanism helps to reduce the need for frequent DPT lookups by storing the results of previous checks.

The situations in which the result of a lookup is permitted to be cached in a DPT TLB are:

Successful ATS Translation Completion

When an ATS Translation Request results in a successful ATS Translation Completion with any permissions other than both read and write permissions being denied (R == W == 0), the SMMU is permitted to create a DPT TLB entry. This entry is based on the final enabled stage of translation that the request was subject to.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Successful DPT Walk Without a Lookup Fault

When a walk of the DPT does not result in a DPT lookup fault, and the DPT information returned by the walk does not indicate “No Access”, the SMMU is permitted to create a DPT TLB entry based on the result of the DPT lookup.

This cached entry includes information such as access permissions and VMID, which are used to quickly resolve future access requests to the same memory region.

To ensure the DPT cache contains up-to-date information, there are mechanisms in place for invalidating outdated or incorrect cache entries. This is crucial for maintaining the integrity and security of memory access permissions. The DPT maintenance commands are designed to remove cached DPT information from DPT TLBs:

CMD_DPTI_ALL

Invalidate all cached DPT information within a given security state. This command ensures that any changes in DPT configurations or permissions are enforced by clearing outdated entries from the cache.

CMD_DPTI_PA

For more targeted invalidation, use this command. It enables the invalidation of cached copies of DPT information for a specific granule of a non-contiguous Level 1 entry. This is useful for fine-grained control over DPT cache entries.

6.6.5 DPT lookup errors

The DPT lookup errors are categorized into two main classes, each with specific conditions and reporting mechanisms:

DPT Lookup Process Succeeds but Does Not Grant Access

This condition occurs when the DPT lookup process completes successfully but the access permissions do not allow the transaction being checked. This is a Device Access Fault and is reported as `F_TRANSL_FORBIDDEN`.

DPT Lookup Process Fails

This is known as a DPT lookup fault. The following table shows the conditions that trigger a DPT lookup fault, along with their fault reporting priorities:

Priority	Condition description	Reported as
1	`DPT_WALK_EN = 0`	`DPT_DISABLED`
2	Invalid DPT register configuration	`DPT_WALK_FAULT`
3	GPC fault on level 0 fetch	`DPT_GPC_FAULT`
4	External abort on level 0 fetch	`DPT_EABT`
5	Invalid level 0 descriptor	`DPT_WALK_FAULT`
6	GPC fault on level 1 fetch	`DPT_GPC_FAULT`
7	External abort on level 1 fetch	`DPT_EABT`
8	Invalid level 1 descriptor	`DPT_WALK_FAULT`

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.7 Memory System Resource Partitioning and Monitoring (MPAM)

For MPAM attribute assignment, an SMMU that supports MPAM and implements the Realm programming interface is considered a four-space MPAM component.

By default, both SMMU-originated and client-originated accesses for Realm state use the Realm PARTID space. However, if `SMMU_R_MPAMIDR.HAS_MPAM_NS == 1`, some Realm state accesses might use the Non-secure PARTID space, depending on the configuration of `SMMU_R_GMPAM.MPAM_NS` or `STE.MPAM_NS`, as applicable.

For NoStreamID accesses and the resulting GPT walks, the `MPAM_SP` value used is an **IMPLEMENTATION DEFINED** choice. It might be either:

- The `MPAM_SP` value provided by the device
- The `MPAM_SP` value associated with the target physical address space of the access

The way of assigning PARTID and PMG for client transactions and SMMU-originated transactions for Realm state is the same as that for Non-secure state.

DPT lookups use the MPAM PARTID and PMG values specified in the STE associated with the StreamID. Any GPT access is made with MPAM PARTID and PMG values of the client-originated or SMMU-originated access that caused the GPT access.

6.8 Performance Monitoring Counter Group enhancements

The SMMU for RME DA introduces updates to Performance Monitoring Counter Group (PMCG) to support the Realm security state. The enhancements include:

- Event tracking for DPT lookups, enabling finer-grained visibility into memory translation behavior
- StreamID-based filtering controls for monitoring Realm-specific transactions
- Support for MPAM PARTID and PMG filtering for the Realm PARTID space
- Mechanisms to observe non-attributable events, which are not directly linked to a specific security state
- Event tracking for for NoStreamID devices

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.8.1 Counting events related to DPT lookups

PMCG now supports counting of DPT-related events for components that implement DPT checks. The following table shows architected event IDs which count DPT lookups:

Event ID	Description
2	TLB miss due to incoming transaction or translation request
4	Translation table walk accesses

6.8.2 StreamID and Security State Filtering

For event types that can be filtered on StreamID, the filtering mode which is based on StreamID and SEC_SID is determined by `SMMU_PMCG_EVTYPERn.{FILTER_SEC_SID, FILTER_SID_SPAN}` and `SMMU_PMCG_SMRn.STREAMID`. The following table shows distinct filtering modes for StreamID and SEC_SID matching:

Mode	Description
ExactSID	The filter matches an exact StreamID in one Security namespace
PartialSID	The filter matches a span of StreamIDs in one Security namespace
AllSIDOneSECSID	The filter matches all the StreamIDs in one Security namespace
AllSIDManySECSID	The filter matches all the StreamIDs in many Security namespaces

In `ExactSID`, `PartialSID` and `AllSIDOneSECSID` modes, StreamID filtering is applied and SEC_SID filtering is applied as follows:

- Count only Non-secure events
- Count only Secure events
- Count only Realm events

In `AllSIDManySECSID` mode, StreamID filtering is not applied and SEC_SID filtering is applied. The following table shows the combinations of events that can be counted by SEC_SID filtering:

Combination #	Non-secure events	Secure events	Realm events	NoStreamID Accesses to Root PA
1	Y	N	N	N
2	Y	Y	N	N
3	Y	N	Y	N
4	Y	Y	Y	N
5	Y	N	N	Y
6	Y	Y	N	Y
7	Y	N	Y	Y
8	Y	Y	Y	Y

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

6.8.3 MPAM PARTID and PMG filtering

The existing bit `SMMU_PMCG_EVTYPER<n>.FILTER_MPAM_NS` is removed, and instead a 2-bit field, `FILTER_MPAM_SP`, is introduced. This enables the Realm PARTID space to be specified in addition to the Non-secure PARTID space and Secure PARTID space.

6.8.4 Counting of non-attributable events

A non-attributable event is an event that is not directly associated with a single Security state, but might reveal information about a Security state. None of the architected SMMUv3 events are non-attributable events.

If the Realm programming interface is present, then non-attributable events are only counted if all of the following are true:

- `SMMU_PMCG_ROOTCR.NAO` is 1.
- Either or both of `SMMU_PMCG_SCR.SO` and `SMMU_PMCG_SCR.NAO` are 1.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7. Memory Encryption Contexts extension

Memory Encryption Contexts (MEC) extends the existing support for memory encryption, enabling multiple encryption contexts in the Realm Physical Address Space (PAS) for assignment to Realm virtual machines. The Non-secure, Secure, and Root PASs each have one encryption context.

7.1 MECIDs

Memory Encryption Context IDs (MECIDs) associate a memory access with a memory encryption context. A MECID value is not a system-global identifier. It must be qualified by a PAS. Each PAS has a default MECID value zero.

Each MECID is bound to a cryptographic context for encrypted memory locations. A single MECID can be bound to different cryptographic contexts for different resources.

7.2 MECID allocation

The Non-secure, Secure and Root PASes do not support multiple MECIDs. Those memory accesses are associated with a default MECID value 0.

The Realm PAS supports multiple MECIDs.

The MECIDs are configured in System registers for each supported execution context and translation regime.

MECID register	Comments
MECID_RL_A_EL3	Alternate MECID for EL3 stage 1 translation regime
MECID_P0_EL2	Primary MECID for EL2 and EL2&0 translation regimes
MECID_A0_EL2	Alternate MECID for EL2 and EL2&0 translation regimes
MECID_P1_EL2	Primary MECID for EL2&0 translation regimes
MECID_A1_EL2	Alternate MECID for EL2&0 translation regimes
VMECID_P_EL2	Primary MECID for EL1&0 stage 2 translation regime
VMECID_A_EL2	Alternate MECID for EL1&0 stage 2 translation regime

7.2.1 Effect of MEC on PAS

On the EL3 translation regime, software can only access the Realm PA space using translated addresses, {NSE, NS} fields within the Block or Page descriptors must equal {1, 1} to access the Realm PA space. MECID_RL_A_EL3 defines MECID value.

On the Realm EL2 and Realm EL2&0 translation regimes, TCR_EL2.A1 bit chooses MECID_P1_EL2 or MECID_P0_EL2 for MMU translation table lookup. For translated addresses, the Alternate

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

MECID (AMEC) field on Translation Table descriptor selects whether to use the Primary MECID register value or the Alternate MECID register value. TCR2_EL2 adds AMEC0 and AMEC1 bits to enable the safe update of MECID_A0_EL2 and MECID_A1_EL2 registers. For example, when TCR2_EL2.AMEC0 is `0b0`, use of Block or Page descriptor containing AMEC =1 generates a Translation fault.

On the Realm EL1&0 stage 1 translation, if HCR_EL2.VM is 0, all Realm EL1&0 Realm PAS uses Primary MECID register VMECID_P_EL2. If HCR_EL2.VM is 1, all MMU table lookups use VMECID_P_EL2. For the Realm EL1&0 stage 2 translation address to Realm PAS, the MECID register VMECID_P_EL2 or VMECID_A_EL2 is chosen by AMEC field in Realm Stage 2 translation table descriptors.

7.3 MECID width

The range of MECID width is 1-16 bits. MECID width is a discoverable **IMPLEMENTATION DEFINED** value for this PE, from the MECIDR_EL2 register.

Arm strongly recommends that all components within a Arm system support the same MECID width.

7.4 MECID mismatch

MECID mismatch occurs within a cache, when the MECID associated with a memory access to a location is different to the MECID associated with a copy of the location in the cache.

A mismatch between the active MECID and the target PA's MECID results in a translation fault or data abort. Optionally, the cache line can be scrubbed. This enforces strong isolation between contexts.

7.5 Memory protection engine (MPE)

MPE provides external memory encryption and integrity services as required by Arm CCA security model. Data is encrypted before being written to external memory, and decrypted on reads from external memory. In a system with MEC, MECIDs identify encryption contexts such as encryption keys or tweaks, which might be stored in MECID-indexed tables or MECID-tagged caches.

MPE in an RME system is either configured by Monitor Security Domain (MSD) EL3 firmware or a Trusted subsystem , which both can access resources in the Root PAS. You can also implement MPE as an autonomous system security hardware component.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

7.6 Memory encryption block size

Memory encryption operates on fixed-size blocks,typically 64 or 128 bytes. All protected regions must be aligned to block boundaries; overlapping contexts within a block cause faults.

7.7 MECID on SMMU for RME DA

The MECID facilitates secure differentiation of encryption contexts within systems utilizing memory encryption. It ensures that data encrypted under a specific context cannot be meaningfully accessed or decrypted by devices operating within another encryption context. In systems employing SMMU, MECIDs enforce rigorous isolation between different Realm memory regions, typically associated with individual Realm.

Each Realm is assigned a unique MECID, which is then associated with all memory transactions within that Realm. The determination of MECID value depends on the transaction source:

- For accesses originated by an SMMU, or by devices when translation is enabled, the MECID is specified by the Stream Table Entry's MECID field.
- For transactions where translation is disabled where there are NoStreamID accesses, the MECID is derived directly from the `AWMECID` or `ARMECID` signals.

By associating transactions with their respective MECIDs, the MPE ensures data is encrypted uniquely for each Realm. This effectively prevents an attacker who has compromised encryption within one Realm from decrypting data belonging to another, so maintaining strict security boundaries between isolated environments.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8. SMMU implementation

This chapter describes how SMMU implementations extend to support the Realm Management Extension (RME). It describes how StreamIDs, PAS encodings, and protocol-specific signals are adapted to include the Realm state, ensuring strict isolation between Non-secure, Secure, and Realm transactions. The chapter also highlights how RME-aware SMMUs manage features such as MPAM, MECID, and PCIe IDE/TDISP extensions to maintain consistent protection across the system.

8.1 Stream security

When multiple client devices share an SMMU, their traffic must carry a unique StreamID to distinguish between sources. Each StreamID is associated with a corresponding security identifier, StreamID Security state (SEC_SID), which specifies the security state of the traffic. The SEC_SID determines the register set used during address translation, ensuring each security state has a distinct view of the physical address space. Consequently, Non-secure client devices cannot view or access Secure physical address spaces.

MMU implementations that support RME also provide support for the Realm security state. Root state, however, is not supported for client devices, because these devices inherently lack Root-level privileges.

8.1.1 ACE-Lite

In the ACE-Lite protocol, the SEC_SID is signaled via:

- `AWMMUSECSID` on the AW channel
- `ARMMUSECSID` on the AR channel

For implementations supporting RME, the encoding of `AxMMUSECSID` is defined as the following table shows:

AxMMUSECSID	Stream security
`0b00`	Non-secure
`0b01`	Secure
`0b10`	Realm
`0b11`	RESERVED

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8.1.2 DTU-TBU

In the DTU-TBU protocol, the SEC_SID is specified through the `SEC_SID` field within the `DTI_TBU_TRANS_REQ` message. For RME-supporting implementations, the following table shows the encoding of this field:

SEC_SID	Stream security
0b00	Non-secure
0b01	Secure
0b10	Realm
0b11	RESERVED

8.1.3 DTI-ATS

The DTI-ATS protocol specifies stream security through the `T` bit of the `DTI_ATS_TRANS_REQ` message:

- T = 0: Non-secure state
- T = 1: Realm state

`DTI_ATS_TRANS_REQ.T` can be set to 1, Realm state, only if both `DTI_ATS_CONDIS_REQ.SUP_T` and `DTI_ATS_CONDIS_ACK.SUP_T` bits were set to 1 during the connection sequence. If either of these fields was not set, `DTI_ATS_TRANS_REQ.T` must be set to 0, Non-secure.

Note

DTI-ATS exclusively supports Non-secure and Realm states. For PCIe, StreamIDs represent the PCIe slot rather than an individual device. Therefore, PCIe StreamIDs must never be trusted with Secure privileges.

8.1.4 LTI

Within the LTI protocol, stream security is signaled by the `LASECSID` signal. For implementations that support RME, the following table shows the encoding of `LASECSID`:

LASECSID	Stream security
0b00	Non-secure
0b01	Secure
0b10	Realm
0b11	RESERVED

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8.2 PAS encoding

The Physical Address Space (PAS) encoding defines the security classification for memory accesses initiated by client devices. Unlike SEC_SID, which specifies the security level of the client device itself, PAS encoding indicates the security level associated specifically with individual memory accesses. For instance, a Secure client device might legitimately access Non-secure memory locations, which contain no sensitive data.

8.2.1 ACE-Lite

In ACE-Lite, PAS encoding is represented using the signals `AWNSE` and `AWPROT[1]` on the write address (AW) channel, and `ARNSE` and `ARPROT[1]` on the read address (AR) channel. The following table shows the encoding of `AxNSE` and `AxPROT[1]`:

AxNSE	AxPROT[1]	PAS encoding
0	0	Secure
0	1	Non-secure
1	0	Root, not used for SMMUs
1	1	Realm

The following restrictions apply for the PAS encoding based on the AxMMUSECSID signal:

- If `AxMMUSECSID` is Non-secure, `AxNSE` and `AxPROT[1]` must indicate Non-secure.
- If `AxMMUSECSID` is Secure, `AxNSE` and `AxPROT[1]` can indicate either Non-secure or Secure.
- If `AxMMUSECSID` is Realm, `AxNSE` and `AxPROT[1]` can indicate either Non-secure or Realm.

8.2.2 DTI-TBU

In DTI-TBU, PAS encoding is communicated via the `NSE` and `NS` fields in the `DTI_TBU_TRANS_REQ` message. The following table shows the encoding of the `NSE` and `NS` fields:

NSE	NS	PAS encoding
0	0	Secure
0	1	Non-secure
1	0	Root, not used for SMMUs
1	1	Realm

The following restrictions apply for the PAS encoding based on the SEC_SID of the client device:

- If SEC_SID is Non-secure, {`NSE`, `NS`} must indicate Non-secure.
- If SEC_SID is Secure, {`NSE`, `NS`} can indicate either Non-secure or Secure.
- If SEC_SID is Realm, {`NSE`, `NS`} can indicate either Non-secure or Realm.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8.2.3 DTI-ATS

PAS encoding in DTI-ATS is indicated using the `TE` bit within the `DTI_ATS_TRANS_RESP` message. The `TE` bit specifies the security space associated with the translated address, enabling the PCIe controller to perform appropriate security attribute checks:

- TE = 0: Non-secure address.
- TE = 1: Realm address.

When `DTI_ATS_TRANS_REQ.T` bit is set to 0, the `TE` bit is Reserved and set to 0.

8.2.4 LTI

For SMMU implementations supporting RME, PAS encoding on the LTI interface is indicated by signals `LANSE` and `LAPROT[1]`. The following table shows the encoding of {`LANSE`,`LAPROT[1]`}:

LANSE	LAPROT[1]	PAS encoding
0	0	Secure
0	1	Non-secure
1	0	Root, not used for SMMUs
1	1	Realm

The following restrictions apply for the PAS encoding based on LASECSID:

- If `LASECSID` is Non-secure, {`LANSE`, `LAPROT[1]`} must indicate Non-secure.
- If `LASECSID` is Secure, {`LANSE`, `LAPROT[1]`} can indicate either Non-secure or Secure.
- If `LASECSID` is Realm, {`LANSE`, `LAPROT[1]`} can indicate either Non-secure or Realm.

8.3 NoStreamID accesses

NoStreamID accesses enable system devices to issue transactions through the SMMU without translation, enabling direct interaction with the physical address space. Unlike Global Bypass and Stream Bypass operations, NoStreamID transactions do not involve attribute conversions. However, GPCs are still performed by the SMMU.

8.3.1 ACE-Lite

The ACE-Lite protocol supports NoStreamID accesses via the `AxMMUVALID` signal, which indicates whether SMMU translation is required. The encoding of the `AxMMUVALID` signal is defined as follows:

- AxMMUVALID = 1: SMMU translation is enabled.
- AxMMUVALID = 0: SMMU translation is bypassed, NoStreamID access.
  - The transaction must specify a valid physical address.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- All other AxMMU* signals become invalid in this mode.

8.3.2 LTI

The LTI protocol supports NoStreamID accesses using the LAMMUV signal, which similarly indicates whether MMU translation is required. The encoding for the LAMMUV signal is defined as follows:

- LAMMUV = 1: SMMU translation is enabled.
- LAMMUV = 0: SMMU translation is bypassed, NoStreamID access.
  - The transaction must specify a valid physical address.
  - Only the LA* transport and control signals remain valid.

In a lookaside integration scenario, memory accesses occur through a separate interface, and no memory read or write data is transferred directly over the LTI interface. Nevertheless, GPC is conducted using information provided by the LTI signals. If GPC fails, the associated memory access is prevented from initiating.

8.4 Memory System Resource Partitioning and Monitoring

Modern systems frequently run multiple applications or virtual machines simultaneously, necessitating mechanisms for fair distribution and monitoring of memory resources. MPAM ensures equitable memory system performance allocation among different software entities. MPAM requires integrated support across multiple system components, including CPUs, interconnects, memory subsystems, and SMMUs.

In SMMUs, MPAM supports internal cache partitioning, resource monitoring, and status reporting. Additionally, MPAM attributes propagate downstream, enabling consistent resource management across the system.

8.4.1 MPAM in ACE-Lite

The ACE-Lite protocol carries MPAM information via the AWMPAM and ARMPAM signals. These signals include:

- PMG: 1 bit
- PARTID: Either 9 or 12 bits, depending on MPAM configuration (MPAM_9_1 or MPAM_12_1)
- Security Indicator:
  - MPAM_NS (1 bit), when RME is not supported.
  - MPAM_SP (2 bits), when RME is supported.

The following table shows the encoding of MPAM_NS and MPAM_SP:

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

MPAM_NS	MPAM security
0b0	Secure
0b1	Non-secure

MPAM_SP	MPAM security
0b00	Secure
0b01	Non-secure
0b10	Root
0b11	Realm

8.4.2 MPAM in DTI-TBU

DTI-TBU protocol conveys MPAM information within the `DTI_TBU_TRANS_RESP` message using the following fields:

- PARTID
- PMG
- Security Indicator: `{MPAMNSE, MPAMNS}`

The following table shows the encoding of `{MPAMNSE,MPAMNS}`:

MPAMNSE	MPAMNS	MPAM security
0	0	Secure
0	1	Non-secure
1	0	Root
1	1	Realm

8.4.3 MPAM in DTI-ATS

DTI-ATS does not support MPAM features.

8.4.4 MPAM in LTI

The LTI protocol encodes MPAM information within the `LRMPAM` signal, comprising:

- PMG: 1 bit
- PARTID: Either 9 or 12 bits, based on MPAM configuration (MPAM_9_1 or MPAM_12_1)
- Security Indicator:
    - MPAM_NS (1 bit), when RME is not supported.
    - MPAM_SP (2 bits), when RME is supported.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

The following table shows the encoding of `MPAM_NS` and `MPAM_SP`:

MPAM_NS	MPAM security
`0b0`	Secure
`0b1`	Non-secure

MPAM_SP	MPAM security
`0b00`	Secure
`0b01`	Non-secure
`0b10`	Root
`0b11`	Realm

8.5 Memory Encryption Context Identifier

In systems that support MEC, the SMMU assigns each Realm stream request with a MECID associated with its corresponding StreamID. This chapter explains how different protocols within an SMMU implementation convey the MECID.

8.5.1 ACE-Lite

In ACE-Lite, MECID signals are conveyed as follows:

- AW Channel: Indicated by the `AWMECID` signal
- AR Channel: Indicated by the `ARMECID` signal

Constraints on `AxMECID` values depend on the indicated PAS:

- If `AxNSE/AxPROT` is Non-secure, Secure, or Root, `AxMECID` must be set to 0.
- If `AxNSE/AxPROT` is Realm, `AxMECID` can have any valid value.

8.5.2 DTI-TBU

In DTI-TBU, MECID is communicated using the `MECID` field within the `DTI_TBU_TRANS_RESPEX` message. This field is valid only when: * `SEC_SID` is set to Realm. * MMU translation is enabled (`MMUV` == 1).

The MECID field must be set to zero for: * Non-secure or Secure requests. * When the resulting PAS is Non-secure.

Translation Buffer Units (TBUs) and the Translation Control Unit (TCU) have fixed data widths in the upstream direction: * If MECID width is 0 or legacy TrustZone mode is enabled, then the data width is 160-bit, otherwise it is 192-bit.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8.5.3 DTI-ATS

MECID is not supported in DTI-ATS.

8.5.4 LTI

In LTI, the MECID is conveyed using the `LRMECID` signal. Constraints on the `LRMECID` values according to the Physical Address Space are as follows:

- If {`LRNSE`, `LRPROT[1]`} indicates Non-secure, Secure, or Root, `LRMECID` must be set to 0.
- If {`LRNSE`, `LRPROT[1]`} indicates Realm, `LRMECID` can have any valid value.

8.6 PCIe IDE interactions

In both cached and lookaside integration modes, the SMMU performs address translations for transactions initiated by PCIe devices. Specific PCIe prefixes correspond directly to features within the SMMU:

- The PCIe RequesterID prefix maps directly to the lower-order bits of the StreamID.
- The PCIe PASID prefix enables subdivision of a PCIe function into smaller, logical units. Within the SMMU, the PASID corresponds to the SubstreamID.
    - The use of PASID is optional; thus, a validity bit explicitly indicates its presence.

Also, the PCIe specification defines an Integrity and Data Encryption (IDE) Translation Layer Packet (TLP) prefix, which includes a 1-bit `T` field indicating the trust level of the transaction. Consequently, a transaction can exist in one of three distinct states:

1. No IDE TLP prefix: The transaction is neither encrypted nor explicitly marked for trust.
2. IDE TLP prefix with T=0: The transaction is encrypted and protected, but it originates from an untrusted source.
3. IDE TLP prefix with T=1: The transaction is encrypted, protected, and originates from a trusted source.

In an Arm-based system, transactions without an IDE TLP prefix or with the IDE TLP prefix having `T=0` are classified as Non-secure.

An SMMU implementation does not differentiate between transactions lacking an IDE TLP prefix and those with an IDE TLP prefix having `T=0`. Transactions with an IDE TLP prefix and `T=1` are classified under the Realm state.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

8.6.1 TDISP XT Extensions

The TDISP eXtended TEE (XT) Extensions introduce an extra `XT` bit, evaluated alongside the existing `T` bit. The combination of `XT` and `T` defines the request type as the following table shows:

XT	T	Meaning
0	0	Non-TEE request; must target non-TEE memory.
0	1	TEE request; can target TEE or non-TEE memory.
1	0	TEE request; must target non-TEE memory.
1	1	TEE request; must target TEE memory.

An SMMU client can present `XT` and `T` directly to the SMMU. The SEC_SID is then derived from the bitwise-OR of `XT` and `T` as the following table shows:

T | XT	SEC_SID
0	Non-secure
1	Realm

Note

As with the original IDE mechanism, the SMMU does not distinguish between the absence of an IDE TLP prefix and the presence of a prefix where `{XT, T} = {0, 0}`.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

9. Debug, trace, and profiling

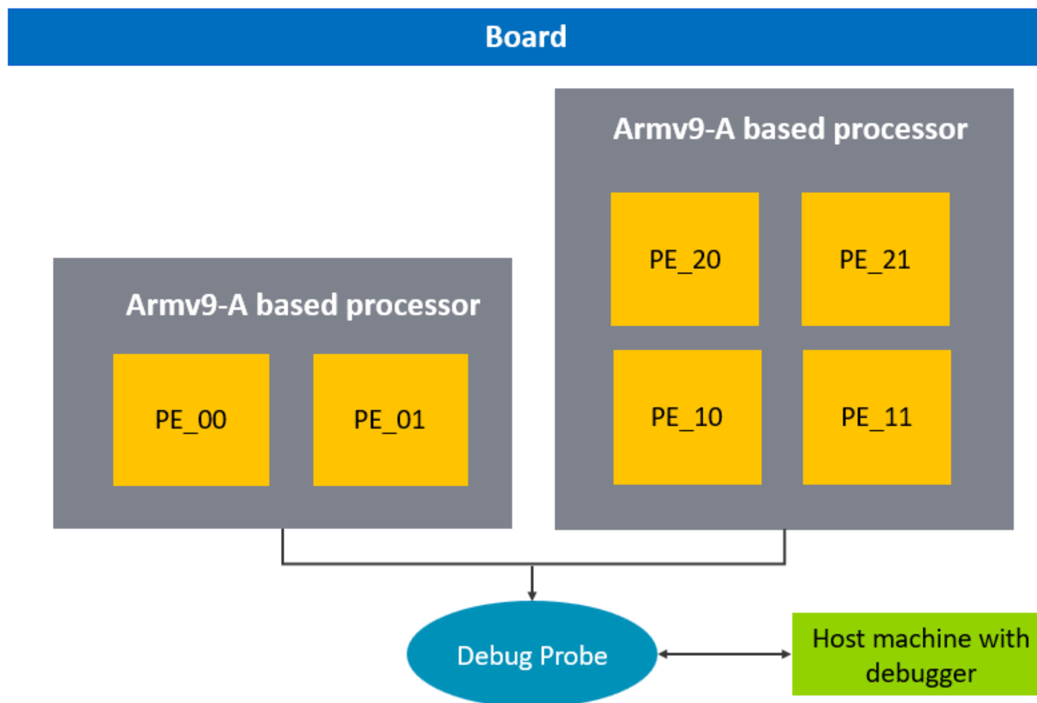
Arm systems include extensive features to support debugging and profiling. We must ensure that these features cannot be used to compromise the security of the system. The Arm architecture, with RME, provides controls to limit which parts of a system can be debugged.

This section assumes familiarity with the base features in Armv9-A and summarizes the changes that RME introduces.

9.1 External debug

External debug is when software is debugged by an agent outside of the processor. The following diagram shows an example of a debug probe that is connected to a development board and a host machine running a debugger:

Figure 9-1: Example of external debug



Signals to enable the different debug, trace, and profiling features help deal with software debugging. The following are separate signals that enable debug in different Security states:

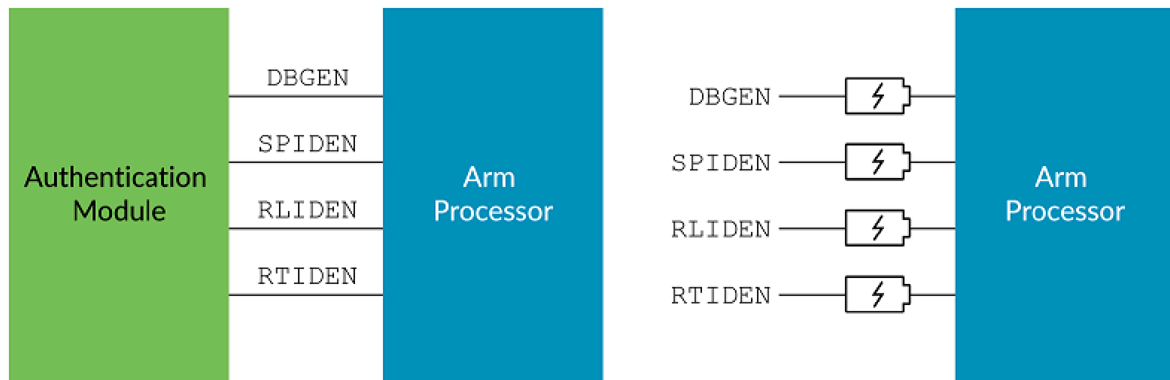
- DBGEN: Top-level invasive debug enable
- SPIDEN: Secure Invasive Debug Enable, controls external ability to debug in Secure state
- RLPIDEN: Realm Invasive Debug Enable, controls external ability to debug in Realm state

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

- RTPIDEN: Root Invasive Debug Enable, controls external ability to debug in Root state

The following figure shows the debug authentication signals that are typically connected to fuses or an authentication module:

Figure 9-2: Debug authentication examples



A silicon vendor uses early development silicon internally. This silicon has the fuses intact, enabling all aspects of the system to be debugged.

Later devices might have a combination of the fuses for RLIDEN, SPIDEN, and RTIDEN blown. This situation enables a development team to only debug specific areas of the system.

Final production silicon in shipping devices has all the fuses blown, to prevent external debug in any state.

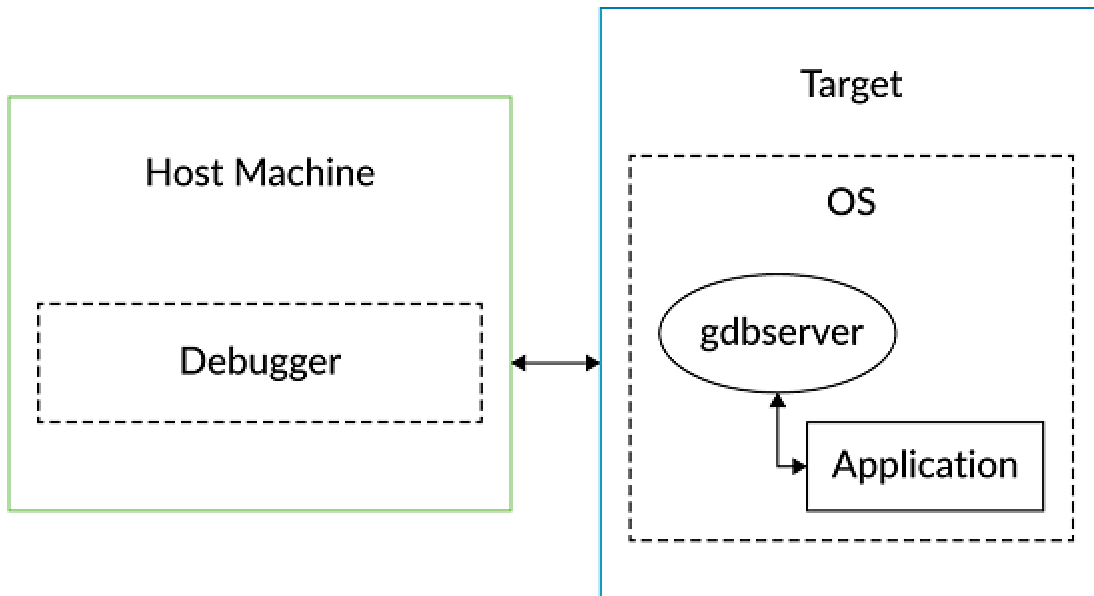
When external debug is disabled for a Security state, for example by blowing a fuse, the processor cannot enter Debug state while in that Security state. An external debugger can still connect to the target. However, the debugger is unable to access the target state or perform run-control until the processor enters a Security state for which external debug is permitted.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

9.2 Self-hosted debug

Self-hosted debug is when software is debugged by an agent running on the same target. The following diagram shows an example of running an application under GDB server:

Figure 9-3: Self-hosted debug



In the diagram, the GDB server is the debug agent running on the target machine. You can connect a graphical debugger to the server. The debugger can also be running on the target or on another machine.

The architectural support for self-hosted debug is available in Realm state. This means that a VM operating within a realm can run a debug server. In this case, the registers that control self-hosted debug are part of the realm context and are saved on realm exit and restored on realm entry.

9.3 Self-hosted trace and SPE

RME introduces minimal changes to the self-hosted trace support and SPE in Armv9-A. Additional fields are added in MDCR_EL3 to control the Security states in which trace and profiling data can be collected.

When used by software in multiple Security states, the registers that control these extensions must be context switched by Exception level 3.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

Both self-hosted trace and SPE use software-defined buffers in memory to hold profiling data. Accesses to the Trace Buffer and Profiling Buffer are subject to Granule Protection Checks. If the check fails, the fault is reported using a Buffer Management Event, instead of an exception. This reporting process is consistent with how VMSA stage 1 and stage 2 faults are handled.

9.4 Performance monitoring

The Performance Monitor Extension (PMU) and Activity Monitor Extension (AMU) are unchanged by RME.

When the PMU is used by software in multiple Security states, the registers that control these extensions must be context switched. This is to avoid the PMU being used to expose information related to one Security state to another.

The AMU provides more limited information and is intended for used for activity monitoring as part of power management. Arm recommends that CNT_CYCLES and CPU_CYCLES are not context switched or disabled when realms are scheduled. Where the AMU provides more counters, access to counters is controlled by Exception level 3.

9.5 Branch Record Buffer Extension

The Armv9.2-A Branch Record Buffer Extension (BRBE) is unchanged by RME. When used by software in multiple Security states, the registers that control BRBE must be context switched.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

10. Related information

Here are some resources that are related to the material in this guide:

- Arm community
- Confidential computing

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential

11. Next steps

This guide introduces the Realm Management Extension (RME), an extension to the Armv9-A architecture.

To learn about the RME in more depth, read the Arm Architecture Reference Manual for A-Profile.

To learn more about the Arm A-profile architecture, read Learn the Architecture guides.

Copyright © 2021, 2024–2025 Arm Limited (or its affiliates). All rights reserved.
Non-Confidential