Arm® Server Base System Architecture 7.2

Platform Design Document

Non-confidential

Document number: DEN0029I

Contents

Release information

Version 7.2 (20 Oct 2024)

- Major additions are Errata for 7.1, and additional Future requirements.
- Move SBSA Appendix A (Support for Secure Firmware) to BSA (818).
- Add RME System Architecture requirements (511).
- Add a reference to RME-DA from the RME System Architecture requirements (680).
- Add a requirement for supporting GIC v4.1 or higher (589).
- Add a requirement for the minimal timer frequency in RS_L8TIME_01 (577).
- Add CXL requirements (790).
- Add CXL Memory ISA requirements (373).
- Specify Cacheability attributes for CXL.cache transactions (665).
- Add UART requirement to Level 3 to coincide with BSA rule relaxation (758).
- Move S_PCIe_09 to Level 3 to coincide with BSA rule relaxation (679).
- Require FEAT_LSE (Atomics) from Armv8.1 for servers (666).
- Recommend a minimum level of per-PE fairness for entropy distribution (539).
- Recommend discrete TPM to be FIFO-based (567).
- Errata for 7.1 - Relax FEAT_NV2 support to conditional requirement (659).
- Errata for 7.1 - Relax AMUv1p1 in Level 7 to a recommendation and require AMUv1 instead (668).
- Errata for 7.1 - Relax PMU_SYS_5 to be a recommendation (607).
- Errata for 7.1 - Relax PMU_EV_07 in Level 7 to make the INT events (80C8, 80C9) recommended not required (678).
- Errata for 7.1 - Relax PCI_ER_02 and PCI_ER_03 in Level 7 to not require MSI/MSI-X for RPs (696).
- Errata for 7.1 - Cleanup MPAM Levels 5 and 7 requirements and relax S_L7MP_06 and RS_L7MP_07 (483).
- Errata for 7.1 - Clean separation between BSA and SBSA (571).
- Errata for 7.1 - S_L4SM_01/S_L4SM_02 should explicitly state if the system must support an SMMU (625).
- Errata for 7.1 - S_L5PE_02 "standard algorithm" errata (581).
- Errata for 7.1 - PAuth2 errata for S_L7PE_06 (582).
- Errata for 7.1 - Update cryptographic requirements for FEAT_PAuth in S_L5PE_02 and S_L7PE_06 (610).
- Errata for 7.1 - Remove ACPI RAS reference from SBSA (557).
- Errata for 7.1 - Update RAS_06 to not conflict with RME system architecture (522).
- Errata for 7.1 - Update S_L7SM_01 SMMU support language to account for RME (636).
- Errata for 7.1 - Clarify S_L3PE_04 (FEAT_LPA and FEAT_LPA2) (669).
- Errata for 7.1 - Clarify the meaning of a Significant Cache for RAS (535).
- Errata for 7.1 - Split S_RAS_01 into S_RAS_01 and a new S_RAS_03, and clarify language around generic counter timebase (295).
- Errata for 7.1 - Update S_L3PE_04 to require a firmware configuration option for a 48bit memory map (629).
- Errata for 7.1 - Fix typos in RS_L8PE_02 (580).
- Errata for 7.1 - Remove S_L6SM_01, which is a duplicate of SMMU_01 (referenced from S_L4SM_03) (657).
- Errata for 7.1 - Change ATS capability discovery to be based on firmware in GPU_04 (594).
- Errata for 7.1 - Update the language on Level 3 Firmware to improve clarity (648).
- Errata for 7.1 - Update PCIe spec reference from 5.0 to 6.0 (587).
- Errata for 7.1 - Replace B_WD_XX with S_L3WD_01 in SBSA L3 checklist (695).
- Errata for 7.1 - Remove S_L7PE_08, S_L7PE_09, S_L7PE_10 recommendations from SBSA L7 checklist (663).
- Errata for 7.1 - Add distinct Rule IDs for Self Hosted Debug rules in Section D (586).
- Errata for 7.1 - Clarify how S_L5GI_01 is different from S_L3GI_01 (749).

- Errata for 7.1 - Remove duplicate text in Section 1.5.5 Watchdog (744).
- Errata for 7.1 - Change "error indication" in SYS_RAS_3 to "poison" (745).
- Errata for 7.1 - Recommend SM3 and SM4 crypto extensions in some markets (738).
- Errata for 7.1 - Remove S_L7PE_03 which is now duplicate of S_L5PE_04 (743).
- Errata for 7.1 - Fix incorrect SERR code in RPCI_ER_07 (672).
- Errata for 7.1 - Relax RPCI_ER_07, RPCI_ER_08 independent error reporting selection for different memory types (693).
- Errata for 7.1 - Correct typo in RS_L6WD_01 (832).

Version 7.1 (06 Oct 2022)

- Major additions are Errata for 7.0, Future requirements, and Appendix D and E.
- SBSA 7.1 version (542).
- Errata for 7.0 - PMU events issues for cache accesses (PMU_EV_02, PMU_EV_03, PMU_EV_04, PMU_EV_10, PMU_EV_11, RPMU_SYS_1, RPMU_SYS_2, RPMU_SYS_3, RPMU_SYS_4) (443).
- Errata for 7.0 - PMU branch events (PMU_EV_03, PMU_EV_10) (538).
- Errata for 7.0 - RAS rules errata (RAS_02, RAS_02A, RAS_07, RAS_08) (320).
- Errata for 7.0 - RAS_01 errata (500).
- Errata for 7.0 - Enhancements to RAS_01 and recommended RAS features for CE counter in self-correcting memory (517).
- Errata for 7.0 - RAS_02 errata (501).
- Errata for 7.0 - RAS_06 clarification (470).
- Errata for 7.0 - SYS_RAS_3 poison requirements seems to cancel each other out (326).
- Errata for 7.0 - S_L7RAS_1 typo (369).
- Errata for 7.0 - S_L5SM_04 typo (426).
- Errata for 7.0 - S_L7TME_2 typo (427).
- Errata for 7.0 - S_L6PE_01 is a duplicate of B_SEC_01-05 (552).
- Errata for 7.0 - Update PCIe terms and abbreviations (464).
- Errata for 7.0 - Add IHI 0091 reference (469).
- Errata for 7.0 - Fix RAS features heading section (472).
- Errata for 7.0 - Fix informative statements that are using "must" (487).
- Errata for 7.0 - Change "Arm recommends" and "Arm Strongly Recommends" terminology (523).
- Errata for 7.0 - Fix invalid rule ID links in the checklist
- SBSA future requirements Checklist (451).
- PE future requirements (378).
- Armv9 self-hosted debug (377).
- SPE PMBIDR_EL1.F (336).
- PBHA not used in PE (406).
- Armv9 requirements with respect to SBSA Levels (459).
- RAS access rule for error nodes on shared resources (257)
- PCIe integration for GPU accelerated compute (440).
- GPU accelerated compute changes (529).
- PCIe MCTP VDM (288).
- PCIe RC external abort on errors (305).
- PCIe I/O Coherency rule (447).
- PCIe Steering Tag handling (424).
- PCIe IDE/CMA recommendation (482).
- PCIe AMBA Integration Guide reference (436).
- CXL related terminology and definitions (468).

Version 7 (31 Jan 2021)

- Major additions are Level-7, Appendix B and C.
- Armv8.6 requirements (248).
- DMA requesters behind SMMU (258).
- Entropy/TRNG requirement (270).
- SMMU-ATS when CXL (273).

- PCIe error handling (279).
- MPAM Level-7 (281).
- PCIe RP EP interoperability (289).
- PCIe miscellaneous rules (287).
- PMU requirements (156).
- RAS requirements (306).
- Branch PMU events (318).
- ERR<n>ADDR.AI bit in RAS (319).
- Errata for 6.0 - Pointer authentication (330).

Version 6.1 (15 Sep 2020)

- The document is reorganized to be a supplement of the Arm BSA document.

Version 6 (16 Sep 2019)

- Armv8.5 requirements (115).
- Instruction to data to instruction coherency (175).
- Nanosecond units for system counter (159).
- Rules for SMMU to ease page sharing between PE and SMMUs (158).
- MSI(-X) must be mapped to LPI (173).
- Mention ServerReady and GIC level 3 clarification (166).
- Clarification on "this" meaning MSI(-X) (103).
- Typo in word address, changed to addresses (102).
- Mislabelling of UARTIMSC in table 15 (101).
- IO virtualization clarifications (112).
- Armv8 RAS extension requirements (133).
- Relaxation of SVE requirement (162).
- Errata clarification on level 5 interrupt controller section of SBSA5.0B (104).
- Level 5 - Section 4.4.2 Interrupt Controller(117).
- SMMU and HTTU support (134).
- SBSA typo on Level 3 PE requirements for SVE (152).
- PCI enumeration related requirements (110).
- PCIe RCiEP and iEP related requirements (108).
- MSI support in SMMU for events (194).

Version 5 (30 May 2018)

- RAS requirements (5).
- ROP and JOB requirements for SBSA (6).
- SBSA and SVE (7).
- Nested virtualization and SBSA (8).
- MPAM requirements for SBSA (9).
- Invisible caches (ban type 2 caches) AKA Forced WB (10).
- Add activity monitor requirements to SBSA (11).
- Crypto requirements to SBSA (12).
- Add support for 48-bit operating systems booting on Armv8.2+ systems with 52-bit PA (13).
- Ban non-standard interrupt controller (14).
- Base frequency standardization (15).
- Assign PPIs for new timers in v8.4 (16).
- Secure EL2 not required (17).
- TLBI range homogeneity (18).
- SVE heterogeneity (19).
- PCIe clarifications (21).
- UART clarifications (22).
- PCIe requirements for assignable devices (23).
- PTM for SBSA-based systems (24).
- PCIe deadlocks (25).

- ACS and Peer-to-peer (26).
- TPM guidance for SBSA (27).
- Deprecate levels 0, 1, and 2 (28).
- Errata - remove references to Prince Algorithm (29).
- Errata - UART trigger levels (30).
- Heterogeneity in server (31).
- Clean to point of persistence support (33).
- Watchdog scaling in SBSA (36).
- Clarify that PEs must be able to access all Non-secure address space (37).
- Appendix I needs to be clearer about RID spaces do not strictly need to supply all 16 bits of width (38).
- Address space input into an SMMU from a device is entirely contiguous, no holes (39).
- Clarify the uniqueness requirements of SPIs that are used to handle legacy PCIe interrupts (A/B/C/D) (40).
- SBSA Armv8.4 and SMMUv3.2 rules on break before make for page tables (41).
- Add reservation for trace buffer overflow PPI (42).
- FP16 support in SBSA (43).
- Correction on stating devices describe their ability to be virtualized through FW (44).
- Clarify IO BAR usage (45).
- Generalize SMMU requirements (46).
- Relaxations in future GIC PPI reservation for Level 5 (79).

Version 3.1 (02 Feb 2016)

- Change of Proprietary Notice. Addition of release history. Non-confidential.

Version 3 (01 Feb 2016)

- Initial Release. Non-confidential.

Arm Non-Confidential Document License (“License”)

This License is a legal agreement between you and Arm Limited (“**Arm**”) for the use of Arm’s intellectual property (including, without limitation, any copyright) embodied in the document accompanying this License (“**Document**”). Arm licenses its intellectual property in the Document to you on condition that you agree to the terms of this License. By using or copying the Document you indicate that you agree to be bound by the terms of this License.

“**Subsidiary**” means any company the majority of whose voting shares is now or hereafter owner or controlled, directly or indirectly, by you. A company shall be a Subsidiary only for the period during which such control exists.

This Document is **NON-CONFIDENTIAL** and any use by you and your Subsidiaries (“Licensee”) is subject to the terms of this License between you and Arm.

Subject to the terms and conditions of this License, Arm hereby grants to Licensee under the intellectual property in the Document owned or controlled by Arm, a non-exclusive, non-transferable, non-sub-licensable, royalty-free, worldwide License to:

(i) use and copy the Document for the purpose of designing and having designed products that comply with the Document;

(ii) manufacture and have manufactured products which have been created under the License granted in (i) above; and

(iii) sell, supply and distribute products which have been created under the License granted in (i) above.

Licensee hereby agrees that the Licenses granted above shall not extend to any portion or function of a product that is not itself compliant with part of the Document.

Except as expressly licensed above, Licensee acquires no right, title or interest in any Arm technology or any intellectual property embodied therein.

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.

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.

THE 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. Arm may make changes to the Document at any time and without notice. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS LICENSE, TO THE FULLEST EXTENT PERMITTED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, IN CONTRACT, TORT OR OTHERWISE, IN CONNECTION WITH THE SUBJECT MATTER OF THIS LICENSE (INCLUDING WITHOUT LIMITATION) (I) LICENSEE’S USE OF THE DOCUMENT; AND (II) THE IMPLEMENTATION OF

THE DOCUMENT IN ANY PRODUCT CREATED BY LICENSEE UNDER THIS LICENSE). THE EXISTENCE OF MORE THAN ONE CLAIM OR SUIT WILL NOT ENLARGE OR EXTEND THE LIMIT. LICENSEE RELEASES ARM FROM ALL OBLIGATIONS, LIABILITY, CLAIMS OR DEMANDS IN EXCESS OF THIS LIMITATION.

This License shall remain in force until terminated by Licensee or by Arm. Without prejudice to any of its other rights, if Licensee is in breach of any of the terms and conditions of this License then Arm may terminate this License immediately upon giving written notice to Licensee. Licensee may terminate this License at any time. Upon termination of this License by Licensee or by Arm, Licensee shall stop using the Document and destroy all copies of the Document in its possession. Upon termination of this License, all terms shall survive except for the License grants.

Any breach of this License by a Subsidiary shall entitle Arm to terminate this License as if you were the party in breach. Any termination of this License shall be effective in respect of all Subsidiaries. Any rights granted to any Subsidiary hereunder shall automatically terminate upon such Subsidiary ceasing to be a Subsidiary.

The Document consists solely of commercial items. Licensee shall be responsible for ensuring that any use, duplication or disclosure of the Document complies fully with any relevant export laws and regulations to assure that the Document or any portion thereof is not exported, directly or indirectly, in violation of such export laws.

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

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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Arm document reference: PRE-21585

version 5.0, March 2024

About this document

Terms and abbreviations

Term	Meaning
ACS	Access Control Services. A set of features intended to ensure that uncontrolled peer-to-peer transaction cannot occur. See [1].
AER	Advanced Error Reporting. A PCIe feature that enables software to isolate and analyze errors with fine granularity. See [1].
Arm ARM	Arm Architecture Reference Manual. See [2].
ATS	Address Translation Services.
Base Server System	A system that is compliant with the Server Base System Architecture.
BBR	Base Boot Requirements. See [3].
BSA	Base System Architecture. See [4].
CMA	PCIe Component Measurement and Authentication. See [1].
Completer	An agent in a computing system that responds to and completes a memory transaction that was initiated by a Requester.
CXL	Compute Express Link. See [5].
DMA	Direct Memory Access.
ECAM	Enhanced Configuration Access Mechanism.
FRU	Field Replaceable Unit.
GIC	Generic Interrupt Controller.
HDM	Host-managed Device Memory. See [5].
I/O coherent	A device is I/O coherent with the PE caches if its transactions snoop the PE caches for cacheable regions of memory. The PE does not snoop the device cache.
IDE	PCIe Integrity and Data Encryption. See [1].
LPI	Locality-specific Peripheral Interrupt (GICv3 [6]).
MEFN	Memory Error Forward Notification. See [5].
MMIO	Memory Mapped Input Output.
MSC	Memory System Component.
NUMA	Non-uniform memory access.
P2P or Peer-to-peer	See PCIe specification [1] for more details.
PCIe Host Bridge (PHB)	See PCIe specification [1] for more details.
PCMO	Persistent Cache Maintenance Operation. A term used for referring to the Arm instructions related to Persistent memory: DC CVAP and DC CVADP. See [2] for more details on these instructions.
PE	Processing Element, as defined in the Arm ARM.

Term	Meaning
PMU	Performance Monitor Unit.
PPI	Private Peripheral Interrupt.
PRI	Page Request Interface.
RCEC	Root Complex Event Collector. See [1].
RCiEP	Root Complex integrated End Point. See [1].
RCRB	Root Complex Register Base. Memory-mapped register space defined by a PCIe or CXL root complex.
Requester	An agent in a computing system that is capable of initiating memory transactions.
Root Complex (RC)	See PCIe specification [1] for more details.
Root Port (RP)	See PCIe specification [1] for more details.
SBSA	Server Base System Architecture.
SGI	Software-Generated Interrupt.
SPI	Shared Peripheral Interrupt.
SR-IOV	Single Root I/O virtualization. This is a method for a PCIe device to be virtualized. See [1].
SVM	Shared Virtual Memory.
System firmware data	System description data structures, for example ACPI or Flattened Device Tree.
VDM	PCI Express Vendor-defined Messages. See [1].
Viral	A hardware-based containment mechanism used in CXL. See [5].
VM	Virtual Machine.

References

This section lists publications by Arm and by third parties.

See Arm Developer (http://developer.arm.com) for access to Arm documentation.

[1] *PCI Express Base Specification Revision 6.0, version 1.0*. PCI-SIG.

[2] *DDI 0487 Arm® Architecture Reference Manual for A-profile architecture*. Arm Ltd.

[3] *DEN 0044 Arm® Base Boot Requirements*. Arm Ltd.

[4] *DEN 0094 Arm® Base System Architecture*. Arm Ltd.

[5] *CXL specification*. Compute Express Link.

[6] *IHI 0069 Arm® Architecture Specification, GIC architecture version 3.0 and version 4.0*. Arm Ltd.

[7] *DEN 0029C Arm® Server Base System Architecture Version 6.0*. Arm Ltd.

[8] *Enhanced Allocation (EA) for Memory and I/O Resources*. PCI-SIG.

[9] *DEN 0068 CoreSight Base System Architecture*. Arm Ltd.

[10] *DDI 0587 Arm® Reliability, Availability and Serviceability (RAS) specification Armv8, for the Armv8-A architecture profile*. Arm Ltd.

[11] *NIST 800-90*. NIST. 'https://csrc.nist.gov/publications/detail/sp/800-90b/final'.

[12] *SBSA ACS*. Arm Ltd. 'https://github.com/ARM-software/sbsa-acs'.

[13] *IHI 0070 Arm® System Memory Management Unit Architecture Specification, SMMU architecture version 3.3*. Arm Ltd.

[14] *DEN 0129 Arm® Realm Management Extension (RME) System Architecture, version A.d*. Arm Ltd.

[15] *DEN 0114 ARM® PCIe AMBA Integration Guide*. Arm Limited.

[16] *IHI 0091 Arm® CoreSight Performance Monitoring Unit Architecture*. Arm Limited.

[17] *IHI 0054 ARM® System Trace Macrocell Programmers Model Architecture*. Arm Limited.

[18] *DDI 0416 Arm® CoreSight Trace Memory Controller Technical Reference Manual*. Arm Limited.

[19] *ARM® Embedded Trace Router Architecture Specification*. Arm Limited.

[20] *ARM® CoreSight System-on-Chip SoC-600 Technical Reference Manual*. Arm Limited.

[21] *IHI 0029 ARM® CoreSight Architecture Specification*. Arm Limited.

[22] *DEN 0034 ARM® Debug and Trace Configuration and Usage Models*. Arm Limited.

Rules-based writing

This specification consists of a set of individual *content items*. A content item is classified as one of the following:

- Declaration
- Rule
- Goal
- Information
- Rationale
- Implementation note
- Software usage

Declarations and Rules are normative statements. An implementation that is compliant with this specification must conform to all Declarations and Rules in this specification that apply to that implementation.

Declarations and Rules must not be read in isolation. Where a particular feature is specified by multiple Declarations and Rules, these are generally grouped into sections and subsections that provide context. Where appropriate, these sections begin with a short introduction.

Arm strongly recommends that implementers read *all* chapters and sections of this document to ensure that an implementation is compliant.

Content items other than Declarations and Rules are informative statements. These are provided as an aid to understanding this specification.

Content item identifiers

A content item may have an associated identifier which is unique among content items in this specification.

After this specification reaches beta status, a given content item has the same identifier across subsequent versions of the specification.

Content item rendering

In this document, a content item is rendered with a token of the following format in the left margin: L_{iiiii}

- *L* is a label that indicates the content class of the content item.
- *iiiii* is the identifier of the content item.

Content item classes

Declaration

A Declaration is a statement that does one or more of the following:

- Introduces a concept
- Introduces a term
- Describes the structure of data
- Describes the encoding of data

A Declaration does not describe behavior.

A Declaration is rendered with the label *D*.

Rule

A Rule is a statement that describes the behavior of a compliant implementation.

A Rule explains what happens in a particular situation.

A Rule does not define concepts or terminology.

A Rule is rendered with the label *R*.

Goal

A Goal is a statement about the purpose of a set of rules.

A Goal explains why a particular feature has been included in the specification.

A Goal is comparable to a “business requirement” or an “emergent property.”

A Goal is intended to be upheld by the logical conjunction of a set of rules.

A Goal is rendered with the label *G*.

Information

An Information statement provides information and guidance as an aid to understanding the specification.

An Information statement is rendered with the label *I*.

Rationale

A Rationale statement explains why the specification was specified in the way it was.

A Rationale statement is rendered with the label *X*.

Implementation note

An Implementation note provides guidance on implementation of the specification.

An Implementation note is rendered with the label *U*.

Software usage

A Software usage statement provides guidance on how software can make use of the features defined by the specification.

A Software usage statement is rendered with the label *S*.

Progressive terminology commitment

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

This document includes terms that can be offensive. We will replace these terms in a future issue of this document. If you find offensive terms in this document, please contact terms@arm.com.

Feedback

Arm welcomes feedback on its documentation.

If you have any comments or suggestions for additions and improvements create a ticket at

https://support.developer.arm.com.

As part of the ticket include:

- The title (Server Base System Architecture).
- The document ID and version (DEN0029I 7.2).
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

1 Server Base System Architecture

This document is a supplement to the Arm Base System Architecture [4].

1.1 Background

The Server Base System Architecture (SBSA) specifies a hardware system architecture for servers that are based on the Arm 64-bit Architecture. Server system software, for example operating systems, hypervisors, and firmware can rely on this standard system architecture. SBSA extends the requirements specified in the Arm BSA [4].

The Server Base System Architecture embeds the notion of levels of functionality. Each level adds functionality better than the previous level. Unless explicitly stated, all specification items that belong to level N apply to levels greater than N.

The list of rules to implement for each SBSA level is available in Section 1.8. All rules of a level, that are described in that level's checklist, are required to be implemented to be compliant to that level.

An implementation is consistent with a level of the Server Base System Architecture if it implements all of the functionality of that level at performance levels that are appropriate for the target uses of that level. This means that all functionality of a level is expected to perform well when exploited by software.

The SBBR recipe in the Arm Base Boot Requirements (Arm BBR) specification [3] describes firmware requirements for an Arm server system. Where this specification refers to system firmware data, it refers to SBBR compliant firmware as specified in the Arm BBR.

1.1.1 SBSA levels and minimum component versions

Table 2 summarizes the minimum architecture versions that map to the rules that are mentioned in an SBSA level.

This table is indicative only. The rules in each level describe the specific features that are required to be compliant to that level. For a checklist of each level's minimum rules, see Section 1.8.

Table 2: SBSA level mapping summary

Level	PE: A profile	SMMU	GIC
3	v8.0	v2 or v3	v3.0
4	v8.3	v3.0	v3.0
5	v8.4	v3.2	v3.0
6	v8.5 or v9.0	v3.2	v3.0
7	v8.6 or v9.1	v3.2	v3.0

Note

Each SBSA level requires a set of PE features. These features were introduced in different revisions of the architecture. Generally, features that are available in version X of the architecture extension can be optionally implemented in version X-1. Some features can be back-ported even further. See the architecture extension documents for more details.

Table 2 shows the version of the architecture extension at which the required features were introduced.

Note

- In earlier revisions of this document, the SBSA checklist Section 1.8 included references to Arm BSA [4] rules. Starting with SBSA 7.2, these direct references are replaced with new or existing SBSA rules. The SBSA levels definition are semantically equivalent to the corresponding levels in earlier revisions of this document.
- The levels from Level 3 through Level 6 that are presented in this document are semantically equivalent to the corresponding levels in SBSA 6.0 [7].

1.2 Level 3

Section 1.8.1 lists the rules to be implemented for Level 3.

$R_{S\_L3\_01}$ The base server system must comply with BSA Level 1 requirements as specified by Section 3.14.1 from Arm BSA [4].

I For a base server system that supports secure firmware, it is recommended to follow the additional set of recommendations specified by “Support for Secure Firmware” section from Arm BSA [4].

1.2.1 PE architecture

$R_{S\_L3PE\_01}$ PEs must support 4KB and 64KB translation granules at stage 1 and stage 2.

$R_{S\_L3PE\_02}$ PEs must implement 16-bit ASID support.

$R_{S\_L3PE\_03}$ PEs must implement AArch64 at all implemented Exception levels.

I Level 3 systems can be implemented using the Armv8.0 revision of the architecture, or higher. FEAT_LPA [2], large Physical Address (PA) and Intermediate Physical Address (IPA) support, expands the maximum physical address width from 48 to 52 bits. Using 52-bit PA requires 64KB translation granules.

$R_{S\_L3PE\_04}$ Server base systems that make use of FEAT_LPA must support a functional system memory map which is wholly contained within the address range from 0 to 2^48-1 (i.e. the first 256TB of physical address space).

U This may be achieved, for example, using a selectable firmware configuration option, or by hiding any mapped resources that exist beyond 256TB address range.

X This is required to support operating systems that do not use the 64KB granule.

I FEAT_LPA2, introduced in Arm v8.7, enables systems to utilize FEAT_LPA for 4KB and 16KB granules. It is recommended that a system that supports FEAT_LPA (for 64KB granules), and supports 4K or 16K granules, should support FEAT_LPA2 as this will prevent the system memory map adjustment described in S_L3PE_04.

I It is consistent with this specification to implement PEs with support for the AArch32 Execution state.

1.2.2 Memory map

$R_{S\_L3MM\_01}$ To enable Non-secure EL2 hypervisors to use a 64KB translation granule at stage 2 MMU translation, the base system must ensure that all memory and peripherals can be mapped using 64KB stage 2 pages and must not require the use of 4KB pages at stage 2.

$R_{S\_L3MM\_02}$ Peripherals that will be assigned to different virtual machines will be situated within different 64KB regions of memory.

1.2.3 Interrupt controller

$R_{S\_L3GI\_01}$ A base server system must implement at least one interrupt controller that is compliant with the GICv3 or higher architecture [6].

$R_{S\_L3GI\_02}$ All MSI and MSI-X targeting hypervisor and operating system software must be mapped to LPI.

I Rules S_L3GI_01 and S_L3GI_02 provide further restrictions on the allowed interrupt controllers as specified in rules B_GIC_01 and B_GIC_02 in the Arm BSA [4].

1.2.4 PPI assignments

$R_{S\_L3PP\_01}$ The Interrupt IDs must be the same as the recommended values specified by B_PPI_01, B_PPI_02, and B_PPI_03 from Arm BSA [4].

1.2.5 System MMU and device assignment

I Armv8.4 introduces TLB Invalidation instructions which apply to a range of input addresses, instead of just to a single address.

$R_{S\_L3SM\_01}$ If PEs that are used by the base system support TLB range instructions, then all OS visible requesters that contain a TLB must support range invalidates. See FEAT_TLBIRANGE in [2].

1.2.6 Watchdog

$R_{S\_L3WD\_01}$ The base server system must implement a Non-secure Generic watchdog as specified by B_WD_01, B_WD_02, B_WD_03, B_WD_04, and B_WD_05 from Arm BSA [4].

1.2.7 Peripheral subsystems

$R_{S\_L3PR\_01}$ The base server system must implement a UART as specified by B_PER_05 in Peripheral subsystems section from Arm BSA [4].

I Either explicitly or via inheritance from the BSA [4], the SBSA has always required a UART to be implemented. This explicit rule coincides with relaxing the BSA requirements.

$R_{S\_PCIe\_09}$ All I/O subsystems that support PCI Express components must support I/O Coherency. See “I/O coherency: memory types and attributes for PCI Express” in [4].

I Either explicitly or via inheritance from the BSA [4], the SBSA has always required I/O subsystems that support PCI Express components to also support I/O Coherency. This explicit rule coincides with relaxing the BSA requirements.

1.3 Level 4

Section 1.8.2 lists the rules to be implemented for Level 4.

1.3.1 PE architecture

In addition to the Level 3 requirements, the following must be true of the PEs in the base server system:

$R_{S\_L4PE\_01}$ All PEs must implement the RAS extension that is introduced in Armv8.2. See FEAT_RAS in [2].

$R_{S\_L4PE\_02}$ If the system contains persistent memory that is exposed to the OS, all PEs must support the clean to point of persistence instruction (DC CVAP). The instruction must be able to perform a clean to the point of persistence for all memory that is exposed as persistent memory to the OS.

$R_{S\_L4PE\_03}$ All PEs must implement FEAT_VMID16.

$R_{S\_L4PE\_04}$ All PEs must implement FEAT_VHE.

1.3.2 System MMU and device assignment

$R_{S\_L4SM\_01}$ A base server system must support Stage 1 System MMU functionality. This must be provided by a System MMU that is compliant with the Arm SMMUv3, or higher, architecture revision.

$R_{S\_L4SM\_02}$ A base server system must support Stage 2 System MMU functionality. This must be provided by a System MMU that is compliant with the Arm SMMUv3, or higher, architecture revision.

X Stage 1 and Stage 2 System MMU functionalities are required for supporting virtualization features such as VM Device Assignment and Shared Virtual Memory (SVM).

$R_{S\_L4SM\_03}$ The integration of the System MMUs must be compliant with with rules SMMU_01 and SMMU_02, as specified in “SMMUv3 integration” section from Arm BSA [4].

1.3.3 Peripheral subsystems

$R_{S\_L4PCI\_1}$ All peripherals that are intended for assignment to a virtual machine or a user space device driver must be based on PCI Express.

$R_{S\_L4PCI\_2}$ There must be no OS observable use of PCIe Enhanced Allocation [8].

1.4 Level 5

Section 1.8.3 lists the rules to be implemented for Level 5.

1.4.1 PE architecture

In addition to the Level 4 requirements, the following must be true of the PEs in the base server system:

$R_{S\_L5PE\_01}$ All PEs must support changing of translation table mapping size (FEAT_BBM) using the Level 1 or Level 2 solution that is proposed in the Armv8.4 extension. Level 2 is recommended. See Section 1.4.3 for the equivalent requirements for the SMMU.

$R_{S\_L5PE\_02}$ All PEs must implement address authentication using one of the standard algorithms defined by the Arm architecture [2], for example, QARMA5, as indicated by ID_AA64ISAR1_EL1.APA. See FEAT_PAuth, FEAT_PACQARMA5 and FEAT_PACQARMA3 in [2].

$R_{S\_L5PE\_03}$ PEs that are based on Armv8.4 must implement the requirements of the CS-BSA combination C [9].

I CS-BSA combination C[9] requires that PEs based on Armv8.x implement the Armv8.4 Self-hosted trace extension. See FEAT_TRF in [2].

I For PEs that are based on Armv9 architecture, see Section C.

$R_{S\_L5PE\_04}$ All PEs must implement the Activity Monitors Extension (AMU) as indicated by ID_AA64PFR0_EL1.AMU==b0001. See FEAT_AMUv1 in [2].

I It is recommended that PEs implement the enhancements to the Activity Monitor Unit as indicated by ID_AA64PFR0_EL1.AMU == b0010. See FEAT_AMUv1p1 in [2].

$R_{S\_L5PE\_05}$ Where trade regulations allow, all PEs must implement cryptography support for SHA3 and SHA512. See FEAT_SHA3 and FEAT_SHA512 in [2].

I It is recommended that FEAT_SM3 and FEAT_SM4 are also supported in hardware aimed at markets where they are used.

$R_{S\_L5PE\_06}$ All PEs must provide support for stage 2 control of memory types and cacheability, as introduced by the Armv8.4 extensions. See FEAT_S2FWB in [2].

$R_{S\_L5PE\_07}$ If PEs implement nested virtualization (FEAT_NV), then they must also implement enhanced nested virtualization, that is provided by HCR_EL2.NV2 and the VNCR_EL2 register. See FEAT_NV2 in [2].

X FEAT_NV2 can help improve the performance of nested virtualization for some use-cases in servers and other applications. It is also considered a requirement for certain virtualization use-cases.

Note

FEAT_NV2 may become a requirement in future versions of this specification.

1.4.2 Interrupt controller

$R_{S\_L5GI\_01}$ All interrupt controllers presented to the operating system software must be GICv3, or higher. Non-standard extensions of GICv3, such as interrupt combining or forwarding engines, that require platform-specific kernel drivers are not permissible.

X Operating system software should not need to carry platform specific drivers for non-standard GIC extensions.

1.4.3 System MMU and device assignment

$R_{S\_L5SM\_01}$ SMMU implementations must be compliant with the Arm SMMUv3.2 architecture revision or higher.

$R_{S\_L5SM\_02}$ SMMU implementations must provide level 1 or level 2 support for translation table resizing.

I It is recommended that the SMMU implements Level 2. If the SMMU implementation provides Level 2, then it is recommended that the PE also provides level 2. arm-arch-rationale:: MPAM architecture requires that all requesters that can access an MPAM-controlled resource, must support passing MPAM ID information.

$R_{S\_L5SM\_03}$ SMMU implementations must comply with the MPAM requirements specified by B_SMMU_11 from Arm BSA [4].

$R_{S\_L5SM\_04}$ SMMU implementations must comply with the requirements specified by B_SMMU_09 and B_SMMU_20 from Arm BSA [4].

1.4.4 Clock and timer subsystem

1.4.4.1 Operating system

$R_{S\_L5TI\_01}$ A system that is compatible with level 5 will implement a generic counter which counts in nanosecond units. This means that, to the operating system, the reported frequency will be 1GHz.

I Systems are permitted to use the counter scaling (FEAT_CNTSC) that is introduced in Armv8.4 as a method to implement this.

1.4.5 Watchdog

I It is recommended that SBSA Level 5 systems use revision 1 of the Generic watchdog. This is because at 1Ghz the watchdog timeout refresh period is limited to just over 4s.

1.4.6 PPI assignments

$R_{S\_L5PP\_01}$ In addition to the PPI assignment specified in Arm BSA [4], the following PPIs are reserved by the SBSA specification:

Table 3: PPI assignments

Interrupt ID	Interrupt	Description
1056-1071	Reserved	Reserved for future SBSA usage.

Interrupt ID	Interrupt	Description
1088-1103	Reserved	Reserved for future SBSA usage.

1.5 Level 6

Section 1.8.4 lists the rules to be implemented for Level 6.

1.5.1 PE architecture

In addition to the Level 5 requirements, the following must be true of the PEs in the base server system:

$R_{S\_L6PE\_01}$ PEs must comply with PE security requirements as specified by B_SEC_01, B_SEC_02, B_SEC_03, B_SEC_04, and B_SEC_05 from Arm BSA [4].

$R_{S\_L6PE\_02}$ PEs must provide support for Branch Target Identification. Support is indicated by register ID_AA64PFR1_EL1.BT== b0001. See FEAT_BTI in [2].

$R_{S\_L6PE\_03}$ PEs must protect against timing faults being used to guess translation table mappings by implementing the TCR_EL1.E0PD0 and TCR_EL1.E0PD1 controls, and the same in TCR_EL2. See FEAT_E0PD in [2]. Support is indicated by ID register ID_AA64MMFR2_EL1.E0PD==b0001.

$R_{S\_L6PE\_04}$ All PEs must implement FEAT_PMUv3p5 [2].

$R_{S\_L6PE\_05}$ Hardware updates to the Access flag and Dirty state in translation tables, as indicated by ID_AA64MMFR1_EL1. HAFDBS = 0b0010, must be supported. See FEAT_HAFDBS in [2].

Note

It is recommended that the base server system removes the need for data cache clean for instruction to data coherency, and instruction invalidation for instruction to data coherency, as indicated by CTR_EL0.IDC == 0b1 and CTR_EL0.DIC == 0b1 respectively.

$R_{S\_L6PE\_06}$ PEs must provide support for enhanced virtualization traps as indicated by ID_AA64MMFR2_EL1.EVT==b0010. See FEAT_EVT in [2].

$R_{S\_L6PE\_07}$ PEs must comply with MTE requirements as specified by B_PE_16 from Arm BSA [4].

$R_{S\_L6PE\_08}$ PEs must comply with SVE requirements as specified by B_PE_17 from Arm BSA [4].

1.5.2 System MMU and device assignment

$R_{S\_L6SM\_02}$ The SMMU must support hardware translation table update (HTTU) of the Access flag and the Dirty state of the page for AArch64 translation tables, as indicated by SMMU_IDR0.HTTU = 0b10.

$R_{S\_L6SM\_03}$ The SMMU must support Message Signaled Interrupts as indicated by SMMU_IDR0.MSI = 0b1.

$R_{S\_L6SM\_04}$ The SMMU must comply with the requirements specified by B_SMMU_03, B_SMMU_04, B_SMMU_05, B_SMMU_13, B_SMMU_14, and B_SMMU_23 from Arm BSA [4].

1.5.3 Watchdog

$R_{S\_L6WD\_01}$ The Architecture version of the Generic watchdog must be v1, that is W_IIDR[19:16] == 0001b.

X In Generic watchdog v0, at 1 GHz the watchdog timeout refresh period is limited to just over 4 seconds.

1.5.4 Armv8 RAS extension requirements

$R_{S\_RAS\_01}$ PEs and other system components that implement the Armv8 RAS extension [10] must use Private Peripheral Interrupts for ERI or FHI if the only interface available for a RAS node is System register based.

$R_{S\_RAS\_03}$ When the RAS Timestamp Extension is implemented by a PE or other system component that implements the Armv8 RAS extension [10], the PE or other system component must use the generic counter time base for the RAS Timestamp, if the generic counter time base is available at the component. When the generic counter time base is used, ERR<n>FR.TS will be either b00 or b01.

I The generic counter time base is available for all PEs in the base server system. For other system components, it is recommended that the generic counter time base is used if the error record timestamps are to be consumed by hypervisor and operating system software. However, it is recognized that this is not always possible for system components that have no other requirement to use the generic counter time base and where the complexity of adding access to the generic counter time base would be prohibitive. For example, the addition of a time synchronization circuit to receive the generic counter time base from the system that is not otherwise needed.

I It is recommended that the system describes the relationship between the generic counter time base and any other counter that might be used in a RAS node.

I It is recommended that the RAS Timestamp Extension is implemented.

I Support for error injection is OPTIONAL, however it is recommended that if error injection is supported, the standard programming model that is described in the Arm RAS System Architecture version 1.1 is followed. Support for error injection is indicated by ERR<n>FR.INJ==b01.

1.5.5 PCIe

In addition to the rules in Arm BSA [4] PCI Express integration section for PCIe root ports and endpoints, the following rules must be implemented.

1.5.5.1 PCIe integration

$R_{S\_L6PCI\_1}$ On-chip peripherals that are presented as PCIe devices comply with the requirements as specified by B_REP_1 and B_IEP_1 from Arm BSA [4].

1.6 Level 7

Section 1.8.5 lists the rules to be implemented for Level 7.

1.6.1 PE architecture

In addition to the Level 6 requirements, the requirements in this section must be true of all the PEs in the base server system.

$R_{S\_L7PE\_01}$ PEs must implement fine-grained trap support as indicated by ID_AA64MMFR0_EL1.FGT== b0001. See FEAT_FGT in [2].

$R_{S\_L7PE\_02}$ PEs must implement the enhanced counter virtualization functionality as indicated by ID_AA64MMFR0_EL1.ECV == b0010. See FEAT_ECV in [2].

$R_{S\_L7PE\_04}$ PEs must implement the Advanced SIMD Int8 matrix multiply extension as indicated by: ID_AA64ISAR1_EL1.I8MM == b0001. See FEAT_I8MM in [2].

$R_{S\_L7PE\_05}$ PEs must implement the BFLOAT16 extension. See FEAT_BF16 in [2].

$R_{S\_L7PE\_06}$ PEs must implement the FEAT_PAuth2, FEAT_FPAC and FEAT_FPACCOMBINE extensions as defined in [2]. For example, when FEAT_PACQARMA5 is implemented then support for these extensions is indicated by ID_AA64ISAR1_EL1.APA == b0101.

$R_{S\_L7PE\_07}$ Support for SVE is OPTIONAL. Where implemented, PEs must implement the SVE Int8 matrix multiply extension as indicated b ID_AA64ZFR0_EL1.I8MM ==b0001 See FEAT_I8MM in [2].

I It is recommended that PEs implement the data gathering hint feature as indicated by ID_AA64ISAR1_EL1.DGH == b001. See FEAT_DGH in [2].

I It is recommended that PEs implement the WFE fine-tuning delay feature as indicated by ID_AA64MMFR0_EL1. TWED == b0001. See FEAT_TWED in [2].

I It is recommended that the enhanced PAN feature is implemented, which is introduced in Arm v8.7-A but can be implemented from v8.1-A. Support for enhanced PAN is indicated by ID_AA64MMFR1_EL1.PAN == b0011.

1.6.1.1 RAS

This section describes the requirements for the Reliability, Availability and Serviceability (RAS) extension. See FEAT_RAS in [2].

$R_{S\_L7RAS\_1}$ For containable errors, error exceptions on reads from Normal memory must be taken as synchronous Data Abort exceptions.

$R_{S\_L7RAS\_2}$ Errors that are signaled to a PE on speculative accesses must not generate Abort exceptions.

I Containable errors are those which have not been silently propagated by the PE and can be taken as a containable error. Other error exceptions are taken as either synchronous data aborts or asynchronous SError interrupts.

1.6.1.2 Transactional Memory Extension

Implementation of Transactional Memory Extension (TME) is OPTIONAL. However, if implemented, the rules in this section apply.

$R_{S\_L7TME\_1}$ All PEs must have the same value of ID_AA64ISAR0_EL1.TME.

$R_{S\_L7TME\_2}$ The latency of starting and committing a transaction must not be higher than the latency of the code sequence that is recommended for acquiring and releasing a spinlock.

$R_{S\_L7TME\_3}$ For adequate performance of applications written in Java and C/C++, hardware must support a read set size of at least 512 objects and a write set size of at least 300 objects, assuming average object size to be 128 bytes.

$R_{S\_L7TME\_4}$ It is recommended that the hardware cache coherency facilities of the processor are used to detect transactional conflicts. This is also known as eager conflict detection.

$R_{S\_L7TME\_5}$ It is recommended that implementations do not generate a transactional conflict when a read generated by a PRFM instruction or by hardware prefetching, accesses a location within the transactional write set of a transaction.

1.6.2 MPAM

$R_{S\_L7MP\_01}$ PEs must implement the MPAM extension. See FEAT_MPAM in [2].

$R_{S\_L7MP\_02}$ PEs must implement a minimum of 16 physical partition IDs and 8 virtual partition IDs.

I It is recommended that the number of partition IDs is derived from the number of PEs in a manner that matches the intended usage of the PARTIDs for the target software. For example, a system targeting cloud hosting might desire to have one PARTID per PE which would correspond to one VM running on one PE at a given time.

$R_{S\_L7MP\_03}$ The implementation must provide MPAM Cache Storage Usage (CSU) monitors and cache portion partitioning for the last-level cache.

I Last level cache refers to an on SoC cache, as opposed to an off-chip DRAM cache.

$R_{S\_L7MP\_04}$ Last-level cache must provide a minimum of 16 Cache Storage Usage monitors.

$R_{S\_L7MP\_05}$ The implementation must provide MPAM Memory Bandwidth Usage monitors (MBWUs) for the interfaces that provide general purpose memory.

I General purpose memory refers to normal DRAM or similar, in contrast to other memory interfaces, for example flash memory or storage class memories.

I It is recommended that the MBWUs implement the MPAM v1.1 64-bit MBWU extension. See FEAT_MPAMv1p1 in [2].

I It is recommended that the MBWUs for an interface be sized so that they can count the total number of bytes that are transferred in a sampling window of time in the order of 10s of milliseconds when operating at maximum interface capacity. This can be achieved for example by:

- Using external capture events, or
- Implementing MPAM v.1.1 64-bit MBWU extension with 44 or 64-bit counters.

I It is recommended that the number of MBWU is derived form the number of PEs in a manner that matches the intended usage of the MBWUs for the target software.

$R_{S\_L7MP\_08}$ The memory map of each Memory-System Component (MSC) that is accessible to Normal world software must be in global address space and have no overlap with other MSCs or peripherals.

1.6.3 Entropy source

$R_{S\_L7ENT\_1}$ To support key and nonce generation, a system must have a hardware entropy source. This source must be a true random number generator that is visible to PE software and meets the requirements that are specified in the NIST SP 800-90 series of specifications [11], or the corresponding national equivalent.

I NIST 800-22 statistical test suite for the validation of random numbers is included in the SBSA Architecture Compliance Suite version 2.4 and higher [12].

1.6.4 SMMU and device assignment

I If an SMMU is in the path of a CXL device [5], the requirements are described in the SMMUv3 Architecture specification [13] chapter called SMMU interactions with CXL.

X The path from a DMA requester to memory must carry the same protection as provided to a PE. An SMMU in the path from DMA requester to memory provides a standard interface to achieve this. S_L7SM_01 enables a server to be deployed without limiting the use-cases involving I/O virtualization.

$R_{S\_L7SM\_01}$ In a base server system, all DMA capable requesters that are visible to the normal world PE software must be behind a Stage 1 and Stage 2 SMMU. This rule applies to types of requesters such as PCIe Root Ports and DMA capable requesters, for example USB, network, disk and accelerators.

Here is an indicative list of the requesters that are exempt from this rule;

- Any on-chip requester whose resources cannot be controlled by PE software, for example system controllers or power management controllers.
- Secure-world requesters.
- Autonomous, DMA capable requesters which are part of the Trusted computing base of the system.

This rule does not apply to SMMUs, interrupt controllers, or debug access ports which are DMA capable, but are explicitly forbidden from being behind an SMMU by design.

I Additionally, in a base server system with RME, all DMA capable requesters must be subject to Granule Protection Checks (GPC), in compliance with [14].

I It is recommended that an ETR [9] is placed behind a SMMU.

$R_{S\_L7SM\_02}$ If there is no SMMU in the path of an ETR, then a CATU as defined in the CoreSight-BSA specification [9] must be implemented for address translation.

I If a CATU is implemented and the device driver for CATU is not present in the OS, trace cannot be captured.

$R_{S\_L7SM\_03}$ SMMU must implement the SMMUv3 Performance Monitors Extension.

$R_{S\_L7SM\_04}$ All SMMU Performance Monitor Counter Groups must implement at least four counters.

I The SMMUv3 Architecture Specification [13] defines which events the System MMU Performance Monitor Extension must implement.

1.6.5 Performance Monitoring Unit

R_{S_L7PMU} The rules in Section A of this specification must be implemented.

1.6.6 System RAS

R_{SYS_RAS} The rules in Section B of this specification must be implemented.

1.6.6.1 Scrubbing

$R_{SYS\_RAS\_1}$ Each NUMA node in the base server system that implements error detection must support patrol scrubbing, supporting both:

- Continuous background scrub of the memory that is connected to the memory controller.
- Targeted scrub that allows run-time configuration of at least:
  - Patrol speed.
  - The region of physical memory connected to the memory controller being scrubbed.

Enabling targeted scrub should not require disabling of background scrub, nor affect the background scrub rate.

I For implemented patrol scrubbing, the following aspects are IMPLEMENTATION DEFINED:

- Programmable features might only be accessible to a system control agent, for example a system control processor, or might be accessible to the PEs.
- The supported patrol speed levels.
- The number of scrubbing channels and whether all channels are programmable.
- The smallest address space granule supported by the scrubbing engine.

It is recommended that control features and IMPLEMENTATION DEFINED capabilities are accessible to an operating system through a common firmware interface, such as the Platform Error Interfaces described in the BBR specification [3].

1.6.6.2 Poison

$R_{SYS\_RAS\_2}$ The system must support the storage and forwarding of poisoned values.

I It is recommended that poison is generated when an uncorrectable data error is detected.

For example, when:

- Corruption is detected in data being evicted from a cache, the cache evicts poison.
- Corruption is detected in data being read from external memory, the memory controller returns poison.
- Corruption is detected in data being received by a component on a write, the error is deferred if possible.
- A poisoned location is modified by a write that does not mask the poison value entirely. In this situation, the location remains poisoned.
  - If the write completely masks the poison, the poison can be removed. In either case, the location is modified and must be treated as a modified location.
- Poison is passed between components in the system with different poison schemes and/or error protection granules. The poison is propagated and expanded if required.

Generating poison can *defer* the error.

U Poison propagation can be achieved in different ways. For example:

1. Setting a poison indicator in the data
2. Carry the error detection bits (for example, ECC, parity, or CRC) with the data

$R_{SYS\_RAS\_3}$ The system should support propagating poison with a transaction (per the definition in [10]) as the transaction propagates through the system.

If a component propagating the transaction is unable to propagate the poison, then the component must record the error as uncorrectable. On reads or non-posted writes, the component must generate one or more in-band error response(s), for example an External Abort or Completer Error. If enabled, an error recovery interrupt would be generated on both posted and non-posted transactions, as described by [10].

All coherent on-chip caches that hold dirty data and coherent addressable memories shall support the ability to retain a previously detected error with the erroneous data.

I It is recommended that a distinct poison indication is saved with the data to avoid the same error being detected again in the future (as opposed to just “leaving bad ECC” with the data in the storage). Examples of a “distinct poison indication” include a separate poison bit or an invalid ECC code.

X The intention of SYS_RAS_3 is to require end-to-end propagation of some error indication between all PEs and all coherent on-chip caches that hold dirty data and coherent addressable memories.

I Details of poisoning schemes, for example, the error protection granule and poison granule sizes, are IMPLEMENTATION DEFINED.

I There is no architecturally-defined method to remove poison from a location, for example by restoring it to a known uncorrupted value. There is no requirement for this type of a method to be possible. It is recommended that if removing poison is supported, systems provide details of the method or a means to execute it in publicly available documentation.

1.6.7 PCIe

In addition to the rules in Arm BSA [4] for PCIe root ports and endpoints, the following rules must be implemented.

1.6.7.1 PCIe integration

$R_{S\_PCIe\_01}$ The system must support the translation of PE writes with all byte enable patterns to PCIe write requests. The translation must be done in compliance with PCIe byte enable rules.

$R_{S\_PCIe\_02}$ The Root Port must support the following:

- 1B and 2B read from Prefetchable and Non-prefetchable address spaces.
- 1B and 2B write to Prefetchable and Non-prefetchable address spaces.

This must hold true for accesses to downstream functions as well as to the Root Port itself. This must hold true even when the read or write address is not DW (4 Byte) aligned.

$R_{S\_PCIe\_03}$ The Root Complex must:

- Send 2B PE writes that are 2B aligned as 2B PCIe writes.
- Send 4B PE writes that are 4B aligned as 4B PCIe writes.
- Send 8B PE writes that are 8B aligned as 8B PCIe writes.

$R_{S\_PCIe\_04}$ The System must ensure that:

- Aligned 2B writes from Endpoints reach the target as 2B writes.
- Aligned 4B writes from Endpoints reach the target as 4B writes.
- Aligned 8B writes from Endpoints reach the target as 8B writes.

1.6.7.2 i-EP

X S_PCIe_0 prevents the Root port PHY LTSSM from waiting indefinitely for the reception of a TS1 ordered set, with the Disable Link bit set before considering its lanes as disabled.

$R_{S\_PCIe\_05}$ Root Port PHY LTSSM must have the capability to consider its lanes as Disabled when all the following conditions are met:

- Root port transmit side has sent TS1 ordered set with Disable bit set
- Root port transmit side has sent the EIOSQ
- Root port receive side has received EIOSQ from the downstream device or switch

See section 4.2.7.9 in [1].

1.6.7.3 Error handling

$R_{PCI\_ER\_01}$ Root Port must support the Advanced Error Reporting feature as described in section 6.2 and 7.8.4 of [1].

I It is recommended that Root Ports implement the MSI or MSI-x capabilities or both. See sections 6.1, 6.2.6 and 6.2.4.2 of [1].

I The Root Port error report includes errors from downstream devices, or error detected within the Root port.

$R_{PCI\_ER\_04}$ The Root Port must log and report the PCIe errors it detects using the AER mechanism. See figure 6-3, section 6.2.6 in [1].

$R_{PCI\_ER\_05}$ The Root Port must implement Downstream Port Containment feature. See section 6.2.10 in [1].

$R_{PCI\_ER\_06}$ The Root Port must comply with the rules stated in the PCIe specification [1] for transaction layer behavior during DPC. See section 2.9.3 in [1].

1.7 Future requirements

The following section lists a preview of new rules that will be required in a future SBSA level, which will be published in a future version of this document.

The list of rules to be implemented for future SBSA level is available in Section 1.8.6.

1.7.1 PE architecture

In addition to the Level 7 requirements, the following must be true of the PEs in the base server system:

$R_{S\_L8PE\_01}$ PEs must implement the XS attribute (FEAT_XS) functionality as indicated by ID_AA64ISAR1_EL1.XS = 0b0001.

$R_{S\_L8PE\_02}$ PEs must implement the support for WFET and WFIT instructions (FEAT_WFxT) functionality as indicated by ID_AA64ISAR2_EL1.WFxT == 0b0010.

$R_{S\_L8PE\_03}$ PEs must implement the support for atomic single-copy 64-byte stores with and without return (FEAT_LS64, FEAT_LS64_V) as indicated by ID_AA64ISAR1.LS64 == 0b0010.

$R_{S\_L8PE\_04}$ PEs must implement the Enhanced PAN support (FEAT_PAN3) as indicated by ID_AA64MMFR1_EL1.PAN == 0b0011.

$R_{S\_L8PE\_05}$ PEs must implement the Armv8.7 enhancements for PMU (FEAT_PMUv3p7) as indicated by ID_DFR0.PerfMon == 0b0111.

$R_{S\_L8PE\_06}$ Implementation of FEAT_BRBE that is introduced in Armv9.2 is OPTIONAL for PEs that are based on Armv9 architecture. However, if FEAT_BRBE is implemented, the branch record buffer must support at least 32 branch records.

$R_{S\_L8PE\_07}$ PEs must not implement functionality for Page-Based Hardware Attributes (PBHA) bits of the VMSAv8-64 block and page descriptors. This is applicable to Stage-1 and Stage-2.

$R_{S\_L8PE\_08}$ PEs must implement FEAT_LSE [2] as required by the rule B_PE_25 in Arm BSA [4].

1.7.2 Realm Management Extensions

The following must be true of server systems that contain PEs that are based on Armv9 architecture:

$R_{S\_L8RME\_1}$ Systems that implement Arm Realm Management Extensions (RME) must be compliant with all the rules defined in version A.d, or a later but compatible version, of the Arm RME System Architecture specification [14]

I Some RME features may require RME System Architecture specification versions beyond A.d. For example, device assignments in Realms is described by the RME-DA rules in version B.a [14].

1.7.3 Self-hosted debug

The following must be true of the PEs in the base server system that are based on Armv9 architecture:

$R_{S\_L8SHD\_1}$ PEs must implement ETE Level-1 as described in Self-hosted Debug Section C.5.3, rules ETE_01, ETE_02, ETE_03, ETE_04, ETE_05, ETE_06, ETE_07, ETE_08, ETE_09, ETE_10.

I Implementation of STM Level-1 as described in Self-hosted Debug, Section C.6, rules STM_01 to STM_29, is OPTIONAL.

1.7.4 System MMU and device assignment

$R_{S\_L8SM\_01}$ SMMU implementations must be compliant with the Arm SMMUv3.3 architecture revision or higher.

1.7.5 System RAS

$R_{SYS\_RAS\_4}$ Resources that are shared by two or more PEs, and are implementing Armv8 RAS Extensions, must minimally support the memory-mapped view of the error nodes.

1.7.6 Clock and timer subsystem

$R_{S\_L8TI\_01}$ The system counter of the Generic Timer must run at a minimum frequency of 50MHz.

I It is recommended that the system counter of the Generic Timer runs at a frequency of 100MHz or more.

1.7.7 Interrupt controller

$R_{S\_L8GI\_01}$ A base server system must implement an interrupt controller that is compliant with the GICv4.1 or higher architecture [6].

1.7.8 PCIe

1.7.8.1 PCIe integration

$R_{S\_PCIe\_06}$ For systems that support MCTP over PCIe VDM, the following rules must be implemented:

- Root Port to Root Port routing of MCTP VDMs must be supported by an SoC if it has more than one Root Port.
  - This rule also applies to the case where a single Root Port is split into multiple Root Ports with smaller link width by bifurcation/quadfurcation.
  - This rule applies even if the Root Ports are in different segments.

$R_{S\_PCIe\_07}$ All observers in the system must observe Inbound writes (for example, writes from downstream devices) with RO=0 in the order in which they were received by the Root Port.

$R_{S\_PCIe\_08}$ All RO=0 writes from a PCIe initiator (that is, under a root-port) must push all previous writes from that initiator to a point of visibility for all observers regardless of the RO value of the prior writes. PCIe write-ordering needs to apply for all observers in the system for requests initiated by PCIe.

I For more information and guidance on integrating PCIe in AMBA AXI based systems, see [15].

$R_{S\_PCIe\_10}$ If Steering Tags are supported by the system, the STE.DCP (Directed Cache Prefetch) control bit in the System MMU must be honored to ensure the feature can be enabled and disabled architecturally.

$R_{S\_PCIe\_11}$ The mechanisms for supporting Steering Tags are IMPLEMENTATION DEFINED, but must have the following properties:

- Steering Tags must be treated as cache allocation hints and must only apply to Normal Cacheable memory transactions in the shareability domain of the I/O Subsystem.
- Steering Tags must not alter coherency guarantees.
- Steering Tags are permitted to be ignored.
- Steering Tag value properties for the platform must be discoverable by software as advertized by firmware.

I A Steering Tag value of 0 indicates no Steering Tag preference and is treated as if no Steering Tag is provided.

I It is recommended that the system support PCIe Integrity and Data Encryption (IDE) and PCIe Component Measurement and Authentication (CMA) to provide security features related to PCIe.

1.7.8.2 Error handling

$R_{PCI\_ER\_07}$ The system must either return all 1s data to the requester PE or trigger a Synchronous External Abort in the requester PE for each of the following cases:

- PE reads targeting MMIO space mapped as Normal memory gets a CA or UR response from the completer or times out (that is, has a Completion Time out (CTO)).
- PE reads targeting MMIO space mapped as Device memory gets a CA or UR response from the completer or times out (that is, has a Completion Time out).
- PE reads targeting configuration space gets a CA response from the completer or times out (that is, has a Completion Time out).
- These requirements apply even if the UR/CA is a response synthesized by the Root Port.
  - The Root Port can synthesize an UR or CA response for outstanding read requests in multiple situations including:
    1. When the Root Port is in Downstream Port Containment (DPC) state (see section 2.9.3 in [1]).
    2. When the Root Port is in DL_Down status (see section 2.9.1 in [1]).
- It is recommended that the selection between returning all 1 data and triggering an External abort is independently configurable for MMIO space mapped as Normal memory, MMIO space mapped as Device memory and Configuration space.

MMIO space can be prefetchable or non-prefetchable memory space.

If an External abort is triggered in response to a PE request because of a PCIe error, and information about the error is logged in an Arm RAS architecture compliant error record, then:

- For CA response, UR response and Completion time out, the SERR code to be logged in the error record is 25.
- UE and ER bits must be set to 1.
- DE and PN bits must be set to 0.
- UET value must be 0b11 (uncorrected error, signalled or recoverable).

Logging the external abort in an Arm RAS Error record does not replace any required logging of the error mandated in the PCIe specification [1], for example, in AER and/or RP-PIO error logs.

$R_{PCI\_ER\_08}$ The system must either return all 1s data or forward the poisoned data to the requester PE for each of the following cases:

- PE reads targeting MMIO space mapped as Normal memory gets poisoned data back from the completer.
- PE reads targeting MMIO space mapped as Device memory gets poisoned data back from the completer.
- PE reads from configuration space gets poisoned data back from the completer.

MMIO space can be prefetchable or non-prefetchable memory space.

The selection between returning all 1s data and forwarding poisoned data must be configurable.

- The default behavior (in the absence of any programming by software/firmware) must be all 1s data return.
- It is recommended that the selection between returning all 1s data and forwarding poisoned data is independently configurable for MMIO space mapped as Normal memory, MMIO space mapped as Device memory and Configuration space.

If the system forwards poisoned response data received from downstream PCIe hierarchy to the requester PE for a read request, and the event is logged in an Arm RAS error record, then the following syndrome must be used.

- The SERR code to be used is 25.
- UE, PN and ER bits must be set to 1.
- DE bit must be set to 0.
- UET value must be 0b11(uncorrected error, signalled or recoverable).

Logging the external abort in an Arm RAS Error record does not replace any required logging of the error mandated in the PCIe specification [1], for example, in AER and/or RP-PIO error logs.

$R_{PCI\_ER\_09}$ The value of “RP Extensions for DPC” bit in the DPC Capability register (see section 7.9.14.2 in [1]) must be configurable by firmware. The OS must still see this bit as Read Only.

$R_{PCI\_ER\_10}$ The Root Port must be capable of triggering DPC if the system returns all 1s data for the following error cases:

- PE reads targeting MMIO space mapped as Normal memory gets a CA or UR response from the completer.
  - If RP-PIO is not implemented or not exposed to the OS, then triggering DPC must be achieved by treating the CA or UR as an un correctable internal error (see section 6.2.9 in [1]).
- PE reads targeting MMIO space mapped as Device memory gets a CA or UR response from the completer.
  - If RP-PIO is not implemented or not exposed to the OS, then triggering DPC must be achieved by treating the CA or UR as an un correctable internal error.
- PE reads targeting configuration space gets a CA response from the completer.
  - If RP-PIO is not implemented or not exposed to the OS, then triggering DPC must be achieved by treating the CA as an un correctable internal error.

I For a posted request from downstream PCIe requesters, the following is recommended:

- If the Root Complex detects that the request’s data payload is corrupted, then:
  - Root Complex should poison the data and forward it to the target if the system supports poisoned data forwarding from the Root Complex.
  - Root Complex should log and report an error if the system does not support poisoned data forwarding from the Root Complex.
- If the Root Complex detects an error that cannot be isolated to the write data, then it should log and report the error.
- The logging and reporting can be through the PCIe error reporting mechanisms or through the Arm RAS architecture specified mechanisms.

1.7.9 Entropy source

I It is recommended that a PE is not starved of entropy within a reasonable time window, even if all other PEs are drawing entropy at the highest possible rate. A reasonable time window is defined as the period between recurrent entropy draws, by a single PE, in any relevant real-world use-case.

I It is recommended that a PE does not see variation in entropy latency/bandwidth regardless of the entropy consumption of other PEs.

X If the previous information statement is disregarded, a PE may observe the available entropy in the system to infer information it should not have access to otherwise.

1.7.10 Peripheral subsystems

I Discrete TPM 2.0 chips have either a FIFO or CRB interface as defined by TCG. If a discrete TPM integrated into a system is directly accessible by Non-secure software, it is recommended that the chip have a FIFO interface.

X CRB-based discrete TPM chips advertise the command and response buffer addresses as the physical address 0x00000000_FED40080. Operating System drivers will map those addresses to interact with the TPM. This will not work on systems unless the TPM is integrated as a memory-mapped device at 0x00000000_FED40000. Until this is resolved with TCG, the recommendation is to use FIFO-based TPMs on Arm system.

1.7.11 GPU accelerated compute

R Systems with GPU accelerated compute must implement the rules in Section D.

1.7.12 CXL

$R_{S\_L8CXL\_1}$ If the system supports CXL [5], it must implement Section E, rules CXL_01, CXL_02, CXL_03, CXL_04, CXL_05, CXL_06, CXL_07, CXL_08, CXL_09, CXL_10, CXL_11, CXL_12, and CXL_13.

1.8 SBSA checklist

This section lists the minimum hardware requirements that are required to install, boot, and run a server operating system on bare-metal or within a virtualization environment.

1.8.1 SBSA Level 3 checklist

Module	Rule ID
Base	S_L3_01
PE	S_L3PE_01
PE	S_L3PE_02
PE	S_L3PE_03
PE	S_L3PE_04
Memory map	S_L3MM_01
Memory map	S_L3MM_02
Interrupt	S_L3GI_01
Interrupt	S_L3GI_02
Interrupt	S_L3PP_01
SMMU	S_L3SM_01
Watchdog	S_L3WD_01
UART	S_L3PR_01
PCIe	S_PCIe_09

1.8.2 SBSA Level 4 checklist

In addition to the SBSA Level 3 rules in Section 1.8.1, the following additional rules are required.

Module	Rule ID
PE	S_L4PE_01
PE	S_L4PE_02
PE	S_L4PE_03
PE	S_L4PE_04
SMMU	S_L4SM_01
SMMU	S_L4SM_02
SMMU	S_L4SM_03
PCIe	S_L4PCI_1
PCIe	S_L4PCI_2

1.8.3 SBSA Level 5 checklist

In addition to the SBSA Level 4 rules in Section 1.8.2, the following additional rules are required.

Module	Rule ID
PE	S_L5PE_01
PE	S_L5PE_02
PE	S_L5PE_03
PE	S_L5PE_04
PE	S_L5PE_05
PE	S_L5PE_06
PE	S_L5PE_07
GIC	S_L5GI_01
SMMU	S_L5SM_01
SMMU	S_L5SM_02
SMMU	S_L5SM_03
SMMU	S_L5SM_04
Timer	S_L5TI_01
Interrupt	S_L5PP_01

1.8.4 SBSA Level 6 checklist

In addition to the SBSA Level 5 rules in Section 1.8.3, the following additional rules are required.

Module	Rule ID
PE	S_L6PE_01
PE	S_L6PE_02
PE	S_L6PE_03
PE	S_L6PE_04
PE	S_L6PE_05
PE	S_L6PE_06
PE	S_L6PE_07
PE	S_L6PE_08
SMMU	S_L6SM_02
SMMU	S_L6SM_03
SMMU	S_L6SM_04
Watchdog	S_L6WD_01
RAS	S_RAS_01
RAS	S_RAS_03
PCIe	S_L6PCI_1

1.8.5 SBSA Level 7 checklist

In addition to the SBSA Level 6 rules in Section 1.8.4, the following additional rules are required.

Module	Rule ID
PE	S_L7PE_01
PE	S_L7PE_02
PE	S_L7PE_04
PE	S_L7PE_05
PE	S_L7PE_06
PE	S_L7PE_07
PE RAS	S_L7RAS_1
PE RAS	S_L7RAS_2
PE TME	S_L7TME_1
PE TME	S_L7TME_2
PE TME	S_L7TME_3
PE TME	S_L7TME_4
PE TME	S_L7TME_5
PE MPAM	S_L7MP_01
PE MPAM	S_L7MP_02

Module	Rule ID
PE MPAM	S_L7MP_03
PE MPAM	S_L7MP_04
PE MPAM	S_L7MP_05
PE MPAM	S_L7MP_08
Entropy	S_L7ENT_1
SMMU	S_L7SM_01
SMMU	S_L7SM_02
SMMU	S_L7SM_03
SMMU	S_L7SM_04
PMU	S_L7PMU
RAS	SYS_RAS
RAS	SYS_RAS_1
RAS	SYS_RAS_2
RAS	SYS_RAS_3
PCIe	S_PCIe_01
PCIe	S_PCIe_02
PCIe	S_PCIe_03
PCIe	S_PCIe_04
PCIe	S_PCIe_05
PCIe	PCI_ER_01
PCIe	PCI_ER_04
PCIe	PCI_ER_05
PCIe	PCI_ER_06

1.8.6 SBSA Future Level checklist

1.8.6.1 Base checklist for SBSA future level

In addition to the SBSA Level 7 rules in Section 1.8.5, the following additional rules are required.

Module	Rule ID
PE	S_L8PE_01
PE	S_L8PE_02
PE	S_L8PE_03
PE	S_L8PE_04
PE	S_L8PE_05
PE	S_L8PE_07

Module	Rule ID
PE	S_L8PE_08
RME	S_L8RME_1
SMMU	S_L8SM_01
RAS	SYS_RAS_4
Timer	S_L8TI_01
GIC	S_L8GI_01
PCIe	S_PCIe_06
PCIe	S_PCIe_07
PCIe	S_PCIe_08
PCIe	S_PCIe_10
PCIe	S_PCIe_11
PCIe	PCI_ER_07
PCIe	PCI_ER_08
PCIe	PCI_ER_09
PCIe	PCI_ER_10
GPU	GPU_01
GPU	GPU_02
GPU	GPU_03
GPU	GPU_04
CXL	S_L8CXL_1

1.8.6.2 Additional checklist for SBSA future level for ARMv9-A profile

In addition to the SBSA Future Level rules in Section 1.8.6.1, the following rules are required for servers with PEs that are based on ARMv9 architecture.

Module	Rule ID
PE	S_L8PE_06
PE	S_L8SHD_1

A Performance Monitoring Unit

Performance monitoring features have two main use models:

- Profiling and software optimization
- Monitoring

Profiling is a tool that is used in software optimization. Performance Monitors are used by profiling. Monitoring is a tool that is used in system operations. Performance Monitors are a hardware feature used in monitoring.

Both approaches typically use sampling to read the PMU.

A.1 Sampling

There are two main sampling models for Hardware Performance Monitors (HPM):

- Time-based sampling:

  Software periodically records values from the HPM and records the location in the program. The changes are tracked over time through phases of software execution. The engineer looks for correlations between phases and recorded events.

- Event-based sampling:

  The location in the program, or some other measurement, is recorded when a *sampled* event occurs. This builds up a statistical model of where events occur.

A.2 PE PMU

The requirements in this section must be met by all the PEs in the base server system.

$R_{PMU\_PE\_01}$ The PE implements the Performance Monitors Extension.

$R_{PMU\_PE\_02}$ The PMU overflow signal from each PE must be wired to a unique PPI interrupt with no intervening logic.

$R_{PMU\_PE\_03}$ Each PE must implement a minimum of six PMU event counters and the PMU cycle counter.

I Performance monitors are required for PE local caches and the last level cache. They are OPTIONAL for intermediate cache levels that are not local to the PE.

I The Performance Monitors Extension requires that at least one of the INST_RETIRED and INST_SPEC events is implemented. It is recommended that the INST_RETIRED event is implemented.

$R_{PMU\_EV\_11}$ PEs in the base server system must either:

- Not implement any multithreaded PMU extension. PMEVTYPER<n>_EL0.MT are RES0. ID_AA64DFR0_EL1.MTPMU == 0b1111.
- Implement the ARMv8.6-MTPMU extension. ID_AA64DFR0_EL1.MTPMU == 0b0001.

R_{PMU_SPE} Implementation of the SPE is optional. However, if SPE is implemented, the following requirements must be met:

1. PEs must support Hardware management of the Access Flag and dirty state for accesses made by the SPE[2], as indicated by PMBIDR_EL1.F == 01b.

- If the SPE implementation samples micro-operations, the implementation provides public documentation of the effect of such sampling on the weighting of instructions in the sample population.
- If the SPE implementation samples the Data Source indicator, the implementation provides public documentation of the mappings of the Data Source values to data sources.

- If the SPE implementation includes IMPLEMENTATION DEFINED packets or IMPLEMENTATION DEFINED events in the events packet, the implementation provides public documentation of these packets and events.

I It is recommended that the SPE is implemented.

Note

PMU_SPE relates to the following from [2]:

- An architecture instruction might create one or more micro-ops at any point in the execution pipeline.
- The definition of a micro-op is IMPLEMENTATION DEFINED.
- An architecture instruction might create more than one micro-op for each instruction.
- A micro-op might also be removed or merged with another micro-op in the execution stream. This means that an architecture instruction might create no micro-ops for an instruction.
- Any arbitrary translation of architecture instructions to an equivalent sequence of micro-ops is permitted.
- In some implementations, the relationship between architecture instructions and micro-ops might vary over time. For example, an instruction that generates two micro-ops is twice as likely to be sampled as an instruction that generates a single micro-op.
- It is recommended that applicable implementation details are provided in freely-usable, machine-readable standard formats.

A.3 PE PMU events

$R_{PMU\_EV\_01}$ PEs in the base server system must implement the PMU events that are shown in the following table to measure IPC:

Number	Mnemonic	Description
0x0008	INST_RETIRED	Instruction architecturally executed
0x0011	CPU_CYCLES	Cycles

$R_{PMU\_EV\_02}$ PEs in the base server system must implement the PMU events that are shown in the following table to measure cache effectiveness, for each PE local cache and the last level cache:

Number	Mnemonic	Description
0x0004	L1D_CACHE	Level 1 data cache access
0x0013	MEM_ACCESS	Data memory access
0x0014	L1I_CACHE	Level 1 instruction cache access
0x0016	L2D_CACHE	Level 2 data cache access
0x0027	L2I_CACHE	Level 2 instruction cache access
0x002B	L3D_CACHE	Level 3 data cache access
0x0036	LL_CACHE_RD	Last Level cache memory read

Number	Mnemonic	Description
0x0037	LL_CACHE_MISS_RD	Last Level cache memory read miss
0x0039	L1D_CACHE_LMISS_RD	Level 1 data cache long-latency read miss
0x0040	L1D_CACHE_RD	Level 1 data cache access, read
0x0050	L2D_CACHE_RD	Level 2 data cache access, read
0x0066	MEM_ACCESS_RD	Data memory access, read
0x00A0	L3D_CACHE_RD	Level 3 data cache access, read
0x4006	L1I_CACHE_LMISS	Level 1 instruction cache long latency miss
0x4009	L2D_CACHE_LMISS_RD	Level 2 data cache long-latency read miss
0x400A	L2I_CACHE_LMISS	Level 2 instruction cache long latency miss
0x400B	L3D_CACHE_LMISS_RD	Level 3 data cache long-latency read miss

L<n>D and L<n>I cache events that are not for PE local caches are not required.

Note

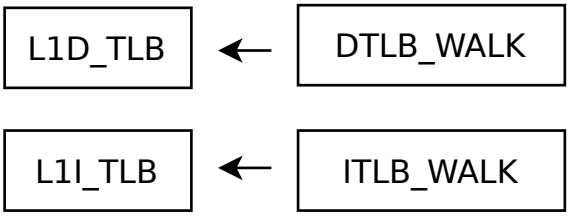
- The definitions of which caches are PE local and which cache is the last level cache are IMPLEMENTATION DEFINED.
- The last level cache usually refers to the last level of cache before memory.
- PE events are attributable to the PE counting the events.

X This is the rationale for the rule PMU_EV_03.

- Effectiveness: For example, a cache is *effective* if accesses to the cache have low latency because they do not miss in the cache.
  - Counting events, for example a cache miss, that indicate effectiveness or ineffectiveness might be more cost efficient than measuring actual latency.
- TLB effectiveness

TLBs are designed to improve performance by caching the results of translation table walks. Therefore, a TLB is not effective if accesses miss in the TLB and cause a translation table walk.

The following events are designed for monitoring effectiveness of the TLBs:



$R_{PMU\_EV\_03}$ The events that are listed in the following table are recommended to be implemented by PEs in the base server system for measuring TLB effectiveness, for each applicable level of PE TLB:

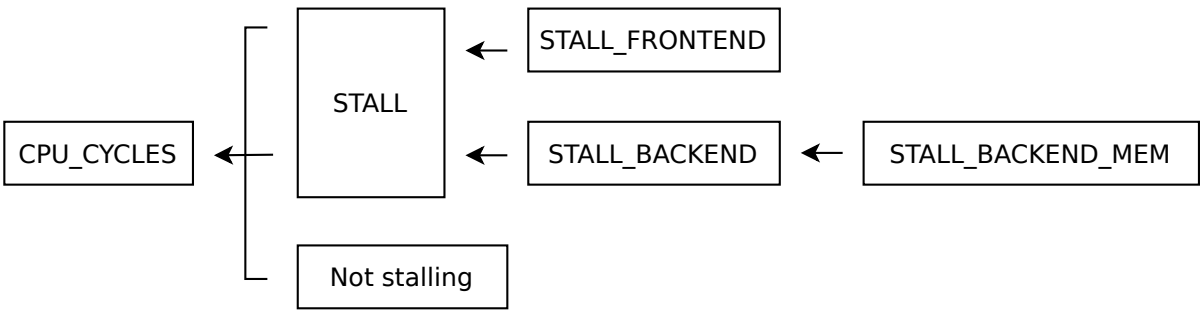
Number	Mnemonic	Description
0x0025	L1D_TLB	Level 1 data TLB access
0x0026	L1I_TLB	Level 1 instruction TLB access
0x0034	DTLB_WALK	Data TLB access with at least one translation table walk
0x0035	ITLB_WALK	Instruction TLB access with at least one translation table walk

$R_{PMU\_EV\_04}$ If the L<n>D and L<n>I cache and TLB events count both demand and non-demand accesses, then the PE must implement additional events that count only demand accesses. It is recommended that following events are defined for this:

Number	Mnemonic	Description
0x8130	L1D_TLB_RW	Level 1 data TLB demand access
0x8131	L1I_TLB_RD	Level 1 instruction TLB demand access
0x813C	DTLB_WALK_RW	Data TLB demand access with at least one translation table walk
0x813D	ITLB_WALK_RD	Instruction TLB demand access with at least one translation table walk
0x8140	L1D_CACHE_RW	Level 1 data cache demand access
0x8141	L1I_CACHE_RD	Level 1 instruction cache demand access
0x8148	L2D_CACHE_RW	Level 2 data cache demand access
0x8149	L2I_CACHE_RD	Level 2 instruction cache demand access
0x8150	L3D_CACHE_RW	Level 3 data cache demand access

However, implementations can also meet this requirement using IMPLEMENTATION DEFINED events. For the purposes of this rule, demand access is as defined for these events in [2].

X The following events are designed for accounting for cycles.



$R_{PMU\_EV\_05}$ The events listed in the following table must be implemented for cycle accounting:

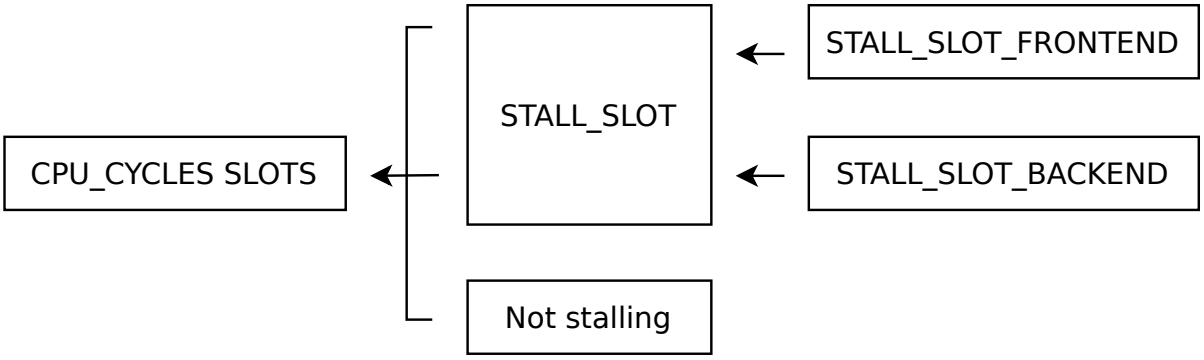
Number	Mnemonic	Description
0x0011	CPU_CYCLES	Cycle
0x0023	STALL_FRONTEND	No operation sent for execution due to the frontend

Number	Mnemonic	Description
0x0024	STALL_BACKEND	No operation sent for execution due to the backend
0x003C	STALL	No operation sent for execution
0x4005	STALL_BACKEND_MEM	Memory stall cycles

A.3.1 Fractional cycle accounting

X This is the rationale for the rule PMU_EV_06.

The following events are designed for accounting for fractional cycles:



The IMPLEMENTATION DEFINED constant SLOTS is discoverable from the system register PMMIR_EL1.SLOTS. It is the maximum number that STALL_SLOT can increment by in a single CPU_CYCLE.

Fractional events are similar in concept to cycle accounting events, but focus on the resources of the CPU. For example, if a CPU can issue up to three instructions in a cycle, then the STALL_SLOT events can count up-to three per cycle.

Note

Another way to view this is that the CPU is being profiled like it issues one instruction per cycle, but operates at three times the frequency.

These resources are referred to as *slots*.

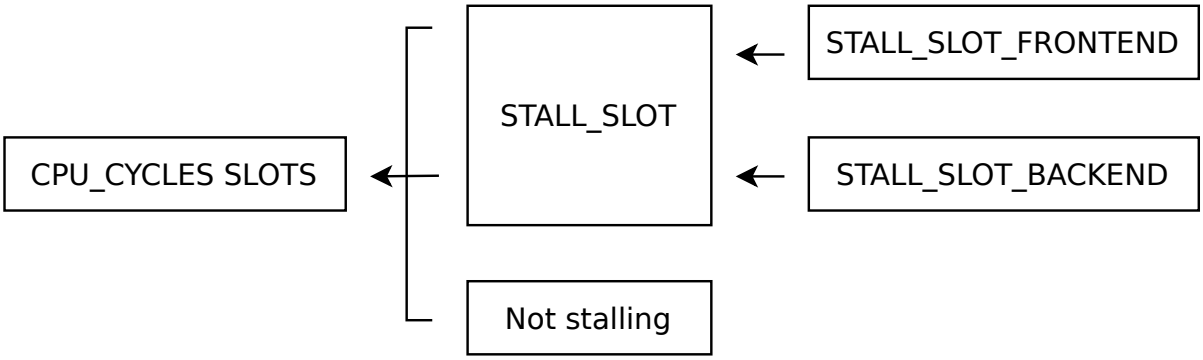
This top-down approach allows for subdivision of these resources, but does not necessarily require strict accounting of resources.

A.3.2 PE utilization

It is useful to break down the operations that are issued for execution into:

- Operations that are architecturally retired
- Operations that only perform wasted work, for example, because of speculation.

The following events are designed for analyzing the impact of speculation:



Here are some measurements to profile the PE utilization.

- Fraction of operations retired

$$f_{\text{retired}} = \frac{\text{OP\_RETIRED}}{\text{OP\_SPEC}}$$

- Operations wasted, for example, due to branch misprediction

$$\text{OP\_WASTED} = \text{OP\_SPEC} - \text{OP\_RETIRED}$$

- Fraction of operations wasted

$$f_{\text{wasted}} = 1 - \frac{\text{OP\_RETIRED}}{\text{OP\_SPEC}}$$

- Utilization of CPU

$$\rho_{\text{CPU}} = \left(1 - \frac{\text{STALL\_SLOT}}{\text{CPU\_CYCLES} \times \text{SLOTS}}\right) \times \left(\frac{\text{OP\_RETIRED}}{\text{OP\_SPEC}}\right)$$

A.3.3 Top-down accounting

Top-down accounting combines the Section A.3.1 and Section A.3.2 methodologies. Together these methodologies allow for four key measurements to be extracted:

$$f_{\text{frontend-bound}} = \frac{\text{STALL\_SLOT\_FRONTEND}}{\text{CPU\_CYCLES} \times \text{SLOTS}}$$
$$f_{\text{backend-bound}} = \frac{\text{STALL\_SLOT\_BACKEND}}{\text{CPU\_CYCLES} \times \text{SLOTS}}$$
$$f_{\text{retired}} = \left(\frac{\text{OP\_RETIRED}}{\text{OP\_SPEC}}\right) \times \left(1 - \frac{\text{STALL\_SLOT}}{\text{CPU\_CYCLES} \times \text{SLOTS}}\right)$$
$$f_{\text{wasted}} = \left(1 - \frac{\text{OP\_RETIRED}}{\text{OP\_SPEC}}\right) \times \left(1 - \frac{\text{STALL\_SLOT}}{\text{CPU\_CYCLES} \times \text{SLOTS}}\right)$$

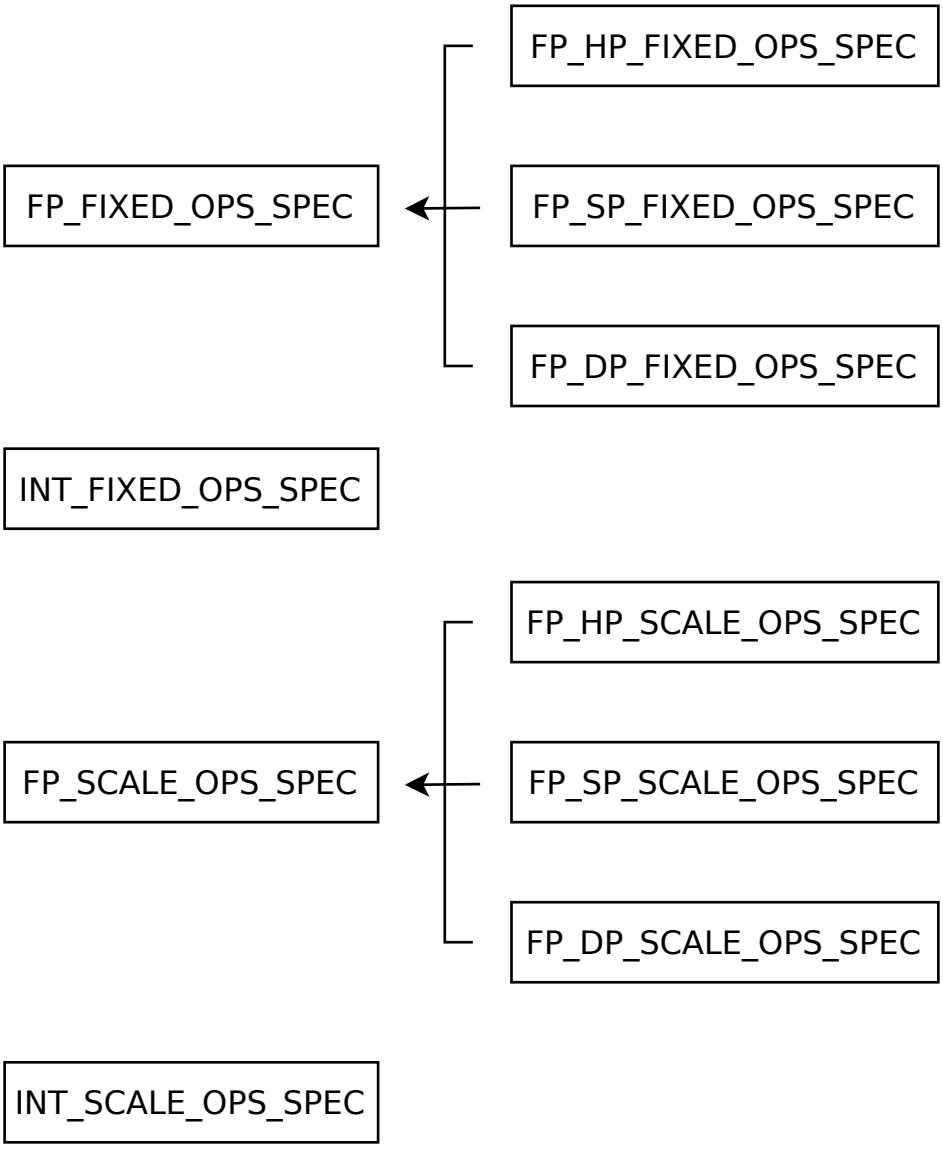
$R_{PMU\_EV\_06}$ The events in the following table must be implemented for Section A.3.3:

Number	Mnemonic	Description
0x0011	CPU_CYCLES	Cycle
0x003A	OP_RETIRED	Micro-operation architecturally executed
0x003B	OP_SPEC	Micro-operation speculatively executed
0x003D	STALL_SLOT_BACKEND	No operation sent for execution on a slot due to the backend
0x003E	STALL_SLOT_FRONTEND	No operation sent for execution on a slot due to the frontend
0x003F	STALL_SLOT	No operation sent for execution on a slot

A.3.4 Workload events (SVE)

X This is the rationale for the rule PMU_EV_07.

SVE [2] defines pairs of events for measuring vector and scalar operation workloads.



These events count the number of operations performed:

- SCALE events count variable-length vector operations. Software must multiply these by the number of 128-bit containers in the vector.
- FIXED events count scalar and fixed-length vector operations, for example Advanced SIMD vector operations.
- For vector operations, the counter increments by an amount that is scaled according to the container (element) size. For example, a single-precision operation on a vector counts twice as many operations as a double-precision operation, because the vector contains twice as many elements.
- Compound operations, for example multiply-accumulate and dot products, count as multiple operations.

Although these events are defined by [2], the FIXED events can be implemented on a PE that does not include [2].

These events allow classification by:

- Integer and floating-point.
- For floating-point only, by data width.

R$_{PMU_EV_07}$ The events listed in the following table must be implemented for measuring workload:

Number	Mnemonic	Description
0x80C1	FP_FIXED_OPS_SPEC	Non-scalable floating-point element operations speculatively executed

If SVE is implemented, the events listed in the following table must be implemented for measuring workload:

Number	Mnemonic	Description
0x80C0	FP_SCALE_OPS_SPEC	Scalable floating-point element operations speculatively executed

I The events listed in the following table are recommended to be implemented for measuring workload:

Number	Mnemonic	Description
0x80C9	INT_FIXED_OPS_SPEC	Non-scalable integer element operations speculatively executed

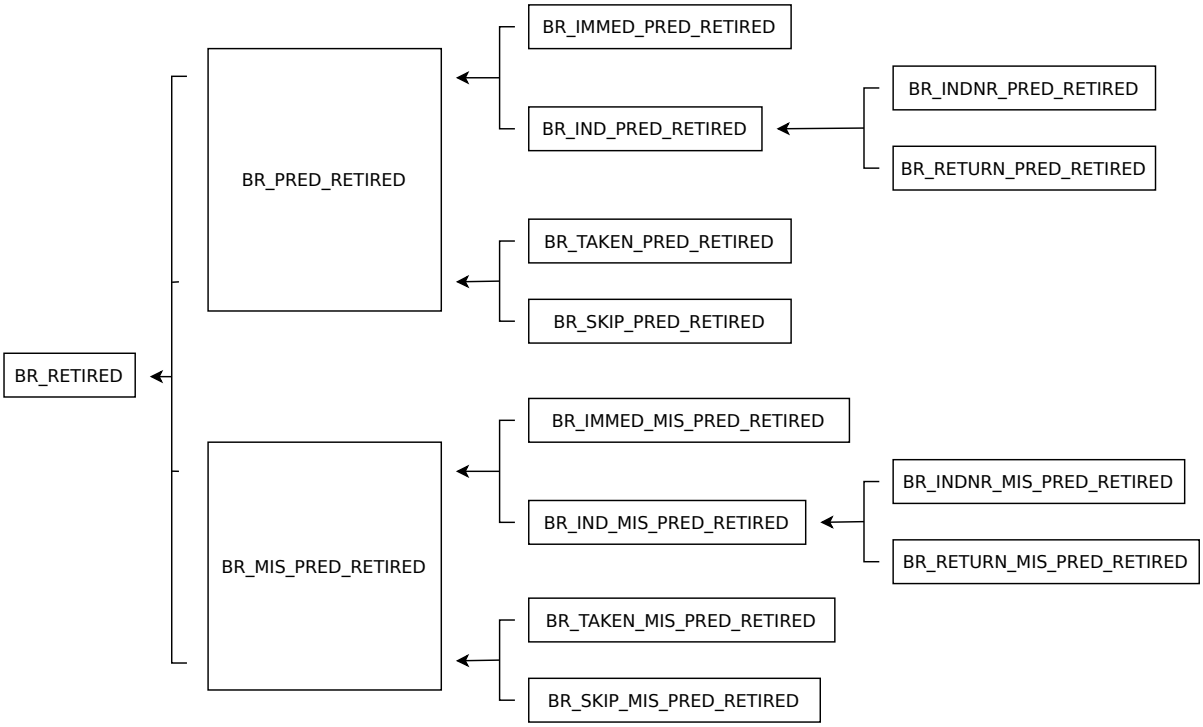
If SVE is implemented, the events listed in the following table are recommended to be implemented for measuring workload:

Number	Mnemonic	Description
0x80C8	INT_SCALE_OPS_SPEC	Scalable integer element operations speculatively executed

A.3.5 Branch predictor effectiveness

X This is the rationale for the rule PMU_EV_08.

The following events are designed for monitoring branch predictor effectiveness.



$R_{PMU\_EV\_08}$ The events listed in the following table are recommended to be implemented for measuring Section A.3.5:

Number	Mnemonic	Description
0x000C	PC_WRITE_RETIRED	Instruction architecturally executed, condition code check pass, software change of the PC.
0x000D	BR_IMMED_RETIRED	Branch instruction architecturally executed, immediate
0x000E	BR_RETURN_RETIRED	Branch instruction architecturally executed, procedure return, taken.
0x0021	BR_RETIRED	Instruction architecturally executed, branch
0x0022	BR_MIS_PRED_RETIRED	Branch instruction architecturally executed, mispredicted
0x8110	BR_IMMED_PRED_RETIRED	Branch instruction architecturally executed, predicted immediate
0x8111	BR_IMMED_MIS_PRED_RETIRED	Branch instruction architecturally executed, mispredicted immediate
0x8112	BR_IND_PRED_RETIRED	Branch instruction architecturally executed, predicted indirect
0x8113	BR_IND_MIS_PRED_RETIRED	Branch instruction architecturally executed, mispredicted indirect

Number	Mnemonic	Description
0x8114	BR_RETURN_PRED_RETIRED	Branch instruction architecturally executed, predicted procedure return
0x8115	BR_RETURN_MIS_PRED_RETIRED	Branch instruction architecturally executed, mispredicted procedure return
0x8116	BR_INDNR_PRED_RETIRED	Branch instruction architecturally executed, predicted indirect excluding procedure return.
0x8117	BR_INDNR_MIS_PRED_RETIRED	Branch instruction architecturally executed, mispredicted indirect excluding procedure return
0x8118	BR_TAKEN_PRED_RETIRED	Branch instruction architecturally executed, predicted branch, taken
0x8119	BR_TAKEN_MIS_PRED_RETIRED	Branch instruction architecturally executed, mispredicted branch, taken
0x811A	BR_SKIP_PRED_RETIRED	Branch instruction architecturally executed, predicted branch, not taken
0x811B	BR_SKIP_MIS_PRED_RETIRED	Branch instruction architecturally executed, mispredicted branch, not taken
0x811C	BR_PRED_RETIRED	Branch instruction architecturally executed, predicted branch
0x811D	BR_IND_RETIRED	Branch instruction architecturally executed, indirect branch
0x811E	BR_INDNR_RETIRED	Branch instruction architecturally executed, indirect excluding procedure return

$R_{PMU\_EV\_09}$ The BR_RETIRED event must count unconditional taken branches.

A.3.6 Latency

X This is the rationale for the rule PMU_EV_10.

The base PMU architecture defines *events* that can be counted, but is also useful to measure the *duration*, or latency, of an event. Arm ARM [2] recommends the following are included as IMPLEMENTATION DEFINED events:

Cumulative occupancy for resource queues, like data access queues, and entry or exit counts, so that average latencies can be determined, separating out counts for key resources that might exist.

Summing cumulative occupancy allows *average* latency to be calculated.

$$\text{Average latency} = \frac{\sum \#\{\text{OCCUPANCY}\}}{\#\{\text{ACCESSES}\}}$$

Armv8.6 does not define a common microarchitectural event for instruction fetches. However, an event for counting instruction fetches, which is analogous to MEM_ACCESS and its associated latency event are recommended.

$R_{PMU\_EV\_10}$ The events listed in the table below are recommended for PEs in the base server system to measure bandwidth, latency, and utilization:

Number	Mnemonic	Description
0x0019	BUS_ACCESS	Bus access
0x0034	DTLB_WALK	Data TLB access with at least one translation table walk
0x0035	ITLB_WALK	Instruction TLB access with at least one translation table walk
0x0060	BUS_ACCESS_RD	Bus access, read
0x0061	BUS_ACCESS_WR	Bus access, write
0x0066	MEM_ACCESS_RD	Data memory access, read
0x0067	MEM_ACCESS_WR	Data memory access, write
0x8120	INST_FETCH_PERCYC	Event in progress, INST_FETCH
0x8121	MEM_ACCESS_RD_PERCYC	Event in progress, INST_FETCH
0x8124	INST_FETCH	Instruction memory access
0x8125	BUS_REQ_RD_PERCYC	Event in progress, BUS_REQ_RD
0x8126	BUS_REQ_WR_PERCYC	Event in progress, BUS_REQ_WR
0x8128	DTLB_WALK_PERCYC	Event in progress, DTLB_WALK
0x8129	ITLB_WALK_PERCYC	Event in progress, ITLB_WALK
0x818D	BUS_REQ_RD	Bus request, read
0x818E	BUS_REQ_WR	Bus request, write

A.3.7 Memory workload

X The rationale for memory workload events is as follows.

The SVE [2] feature defines a pair of vector and scalar memory workload events.

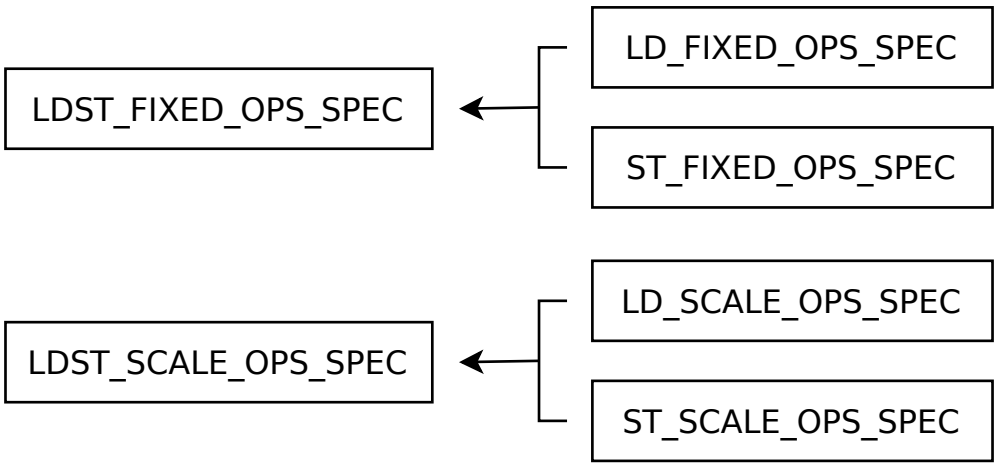


Figure 1: Operation counting

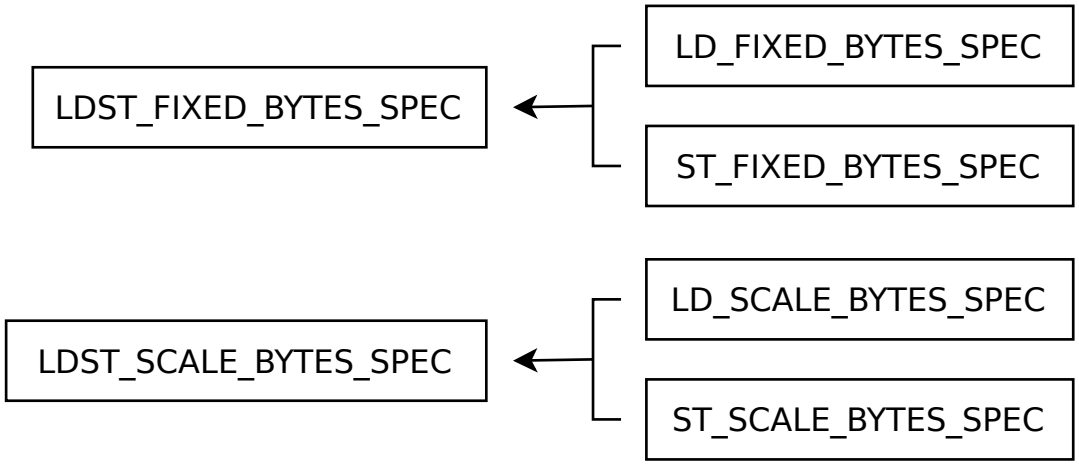


Figure 2: Byte counting

These events can count either the number of operations or the number of bytes transferred:

- SCALE events and FIXED events determine how the counters increment:
  - SCALE events count variable-length vector loads and stores. Software must scale these events by number of 128-bit containers in the vector.
  - FIXED events count scalar and fixed-length vector loads and stores.
- OPS and BYTES determines what the counter counts:
  - For vector loads and stores, OPS events increment by an amount scaled according to the container (element) size.
  - For scalar loads and stores, pairwise load/store OPS events count multiple accesses. All other Load/Store and atomic operations OPS events count a single access.
  - BYTES events count all load/stores, but the counter increments by the number of bytes that are transferred.

Although these events are defined by [2], the FIXED events can be implemented on a PE that does not include SVE.

These events can be used to derive bandwidth performance figures.

I It is recommended that the events listed in the table below are implemented for measuring Section A.3.7:

Number	Mnemonic	Description
0x80CB	LDST_FIXED_OPS_SPEC	Non-scalable load/store element operations speculatively executed
0x80CD	LD_FIXED_OPS_SPEC	Non-scalable load element operations speculatively executed
0x80CF	ST_FIXED_OPS_SPEC	Non-scalable store element operations speculatively executed
0x80DB	LDST_FIXED_BYTES_SPEC	Non-scalable load/store bytes speculatively executed
0x80DD	LD_FIXED_BYTES_SPEC	Non-scalable load bytes speculatively executed
0x80DF	ST_FIXED_BYTES_SPEC	Non-scalable store bytes speculatively executed

I If SVE is implemented, it is recommended that the events listed in the following table are implemented for measuring Section A.3.7:

Number	Mnemonic	Description
0x80CA	LDST_SCALE_OPS_SPEC	Scalable load/store element operations speculatively executed
0x80CC	LD_SCALE_OPS_SPEC	Scalable load element operations speculatively executed
0x80CE	ST_SCALE_OPS_SPEC	Scalable store element operations speculatively executed
0x80DA	LDST_SCALE_BYTES_SPEC	Scalable load/store bytes speculatively executed
0x80DC	LD_SCALE_BYTES_SPEC	Scalable load bytes speculatively executed
0x80DE	ST_SCALE_BYTES_SPEC	Scalable store bytes speculatively executed

I It is recommended that the events in the following table are implemented for refining workload measurements:

Number	Mnemonic	Description
0x80C3	FP_HP_FIXED_OPS_SPEC	Non-scalable half-precision floating-point element operations speculatively executed
0x80C5	FP_SP_FIXED_OPS_SPEC	Non-scalable single-precision floating-point element operations speculatively executed
0x80C7	FP_DP_FIXED_OPS_SPEC	Non-scalable double-precision floating-point element operations speculatively executed

I If SVE is implemented, it is recommended that the events in the following table are implemented for refining workload measurements:

Number	Mnemonic	Description
0x80C2	FP_HP_SCALE_OPS_SPEC	Scalable half-precision floating-point element operations speculatively executed
0x80C4	FP_SP_SCALE_OPS_SPEC	Scalable single-precision floating-point element operations speculatively executed
0x80C6	FP_DP_SCALE_OPS_SPEC	Scalable double-precision floating-point element operations speculatively executed

If half-precision operations are not implemented, then the FP_HP_*_OPS_SPEC events count no operations.

A.4 System performance monitors

I For the purposes of these requirements, an interface is considered to support read and write requests:

- A read is a transaction from a requester to a completer where the completer responds with data.
- A write is a transaction from a requester to a completer where the requester sends data. The completer might acknowledge the request.

Other types of traffic can be supported, but are categorized as reads or writes based on the traffic characteristics.

I Examples of types of traffic that might share an interface include:

- Data transfers
- Cache coherency messages
- Distributed virtual memory messages
- Message-signaled interrupts

I For the purposes of these requirements, for a monitor located on one side of an interface, accesses are either:

- Outbound, that is originating on the same side as the monitor, or
- Inbound, that is originating on the other side of the interface.

I For the purposes of these requirements, an interface is considered to be either unidirectional or bidirectional:

- A unidirectional interface supports requests originating on only one side of the interface. That is, either inbound traffic or outbound traffic, but not both.
- A bidirectional interface supports requests originating on either side of the interface. That is, both inbound and outbound traffic.

I A bidirectional read/write interface therefore supports all of inbound reads, inbound writes, outbound reads, and outbound writes.

The interface between two nodes in a NUMA system is an example of a bidirectional read/write interface.

I Bandwidth is measured in bytes per second. This means that the monitor must be capable of measuring bytes transferred. However, this might be quantized in larger units.

If an interface always transfers larger units, a monitor might count transfers in these units. Either the monitor itself or software using the monitor scales the counted value to bytes transferred.

Furthermore, where an interface usually transfers larger units, a monitor might count transfers in these units, including rounding up smaller transfers to a multiple of the unit. For example, if most traffic to a conventional memory interface is in quantities of 64 bytes, the monitor might present a value that, when scaled, gives an upper-bound for the number of bytes transferred.

However, if the interface often transfers different sized amounts, the monitor should present a reasonably accurate count.

I Total bandwidth is the sum of read bandwidth and write bandwidth. However, monitors should allow monitoring of this type of aggregate events using a minimum set of monitored events. Monitors should not require software to monitor many small, detailed events and compute aggregates.

I Average latency is calculated by accumulating the number of open transactions on each cycle and dividing this by the total number of transactions.

I A single interface might support multiple devices. PCIe is an example of an interface that supports multiple devices on a single interface.

I The measurements that this section describes should be available to at least the host operating system or hypervisor to allow profiling and monitoring of the base server system. Although this section describes these measurements in terms of monitors that perform the measurements, this does not imply any specific approach to generation of the measurements.

I These requirements do not specify how the measurements are generated and collected.

I It is recommended that measurements that are generated by event monitors are implemented in a way that is compatible with the *Arm CoreSight Performance Monitoring Unit Architecture* [16].

$R_{PMU\_BM\_1}$ The base server system must implement bandwidth monitors for each memory interface.

$R_{PMU\_BM\_2}$ The base server system must implement bandwidth monitors for each PCIe interface.

$R_{PMU\_BM\_3}$ The base server system must implement bandwidth monitors for each external accelerator interface.

$R_{PMU\_BM\_4}$ The base server system must implement bandwidth monitors for each chip-to-chip interface.

$R_{PMU\_MEM\_1}$ The base server system must implement average latency monitors for each memory interface.

$R_{PMU\_SYS\_1}$ Each monitor for an interface must be capable of measuring all of the following measurements simultaneously:

- If the interface supports outbound read traffic, average outbound read latency.
- If the interface supports outbound traffic, total outbound bandwidth.
- If the interface supports inbound read traffic, average inbound read latency.
- If the interface supports inbound traffic, total inbound bandwidth.

That is, up to 4 measurements (typically requiring up to 6 event counters) simultaneously.

$R_{PMU\_SYS\_2}$ Each monitor for an interface must be capable of measuring all of the following measurements simultaneously:

- If the interface supports outbound read traffic, outbound read bandwidth and average outbound read latency.
- If the interface supports outbound write traffic, outbound write bandwidth.
- If the interface supports inbound read traffic, inbound read bandwidth and average inbound read latency.
- If the interface supports inbound write traffic, inbound write bandwidth.

That is, up to 6 measurements (typically requiring up to 8 event counters) simultaneously.

I Each monitor that monitors devices on an interface might limit the number of devices that can be monitored simultaneously.

$R_{PMU\_SYS\_5}$ For NUMA systems, it is recommended that each monitor is capable of collecting measurements for each of the following traffic groups simultaneously:

- Local node traffic and remote node traffic.
- All traffic.

$R_{PMU\_SYS\_6}$ For interfaces that carry multiple types of traffic, each monitor must be capable of filtering monitored traffic that is based on its traffic type.

I If FEAT_MPAM is implemented, it is permitted but not required for the bandwidth monitors to be implemented as MPAM bandwidth monitors and support MPAM filtering. Software should be able to monitor each of the measurements for all traffic simultaneously with the requirements to monitor filtered traffic as part of bandwidth partitioning.

$R_{PMU\_SYS\_7}$ Each significant cache in the base server system must be capable of measuring cache effectiveness.

I For example, each significant cache in the system implements hardware performance monitors capable of counting the events listed in the following table:

Mnemonic	Description
CACHE	Cache access.
CACHE_MISS	Cache miss.

Note

Significant cache means any large system cache, for example the last level cache, that might have a significant impact on performance.

These event monitors are in addition to those provided by a PE.

A.4.1 Recommendations

It is recommended that the base server system implement the monitors that are specified in this section.

I It is recommended that the base server system implement average latency monitors for each:

- PCIe interface
- External accelerator interface
- Chip-to-chip interface

I It is recommended that the base server system implement bandwidth monitors for each:

- External PCIe device
- High-bandwidth internal device

I It is recommended that the base server system implement average latency monitors for each:

- External PCIe device
- High-bandwidth internal device

Note

High-bandwidth internal device means any device integrated in the SoC device that is likely to use a significant amount of available bandwidth. Examples might include an integrated GPU, NPU, or other data processing engine. These devices might or might not be PCIe devices.

A.4.1.1 Security

$R_{PMU\_SEC\_1}$ When deployed in production systems, performance monitors must not expose Secure data to untrusted software.

I The definitions of Secure data and untrusted software are IMPLEMENTATION DEFINED and relate to how the information encoded in the data relates to the threat model for the system.

For example, in a typical system that supports both Secure and Non-secure memory, data that is stored in or related to Secure memory is considered Secure data. Other data is considered Non-secure data.

I The above statement requires that the monitor separately measures Secure and Non-secure events, and either ignores Secure events, or does not expose Secure measurements to untrusted software. Alternatively, untrusted software might need to use a firmware or other proxy to access the performance measurements.

B Server RAS

The Arm RAS Extension and Arm RAS System Architecture as described in the Arm RAS for A-profile architecture supplement [10] describe frameworks for RAS features in Arm PE and Arm-based SoC implementations. Neither standard requires any level of RAS features. Both standards are defined to allow scaling from systems where reliability is not a key concern to systems where reliability is a key concern.

B.1 Justifications and impact

This section is informative. The requirements are presented in Section B.2.

For Arm server systems, setting a higher standard of features is recommended, both to ensure greater consistency across Arm implementations and to guide designers of Arm server systems. Arm engages in conversation with cloud vendors on RAS, and the requirements presented in this section of the specification are the result of these discussions.

B.1.1 PE architecture

B.1.1.1 PE error exception handling

The RAS Extension allows error exceptions to be handled in a variety of ways, ranging from:

- Synchronous data abort exceptions. These are precise exceptions that are taken before the corrupt data is consumed by the PE.
- Uncontainable asynchronous SError interrupt exceptions. These might be imprecise exceptions, taken after the corrupt data is consumed and possibly propagated by the PE.

Designers trade off multiple factors when determining how to take error exceptions. If errors occur very late in the execution of an instruction, a power, performance, and area cost might be associated with retaining the state that is required to generate a precise exception.

Nevertheless, having a variety of possible error exceptions creates challenges for portable software. For reliability and availability, the CPU should take exceptions in manners that allow software to contain and possibly recover from an error.

B.1.2 System architecture components

This section includes components of a CPU that the RAS System Architecture section in Arm RAS for A-profile architecture supplement [10] treats as system components, for example caches.

B.1.2.1 Error counters

Some components might generate large numbers of corrected errors if, for example, these components manage a lot of storage. Error counters provide additional information for fault analysis.

Counters should support some form of thresholding. This means that the counter can be configured to generate an event for fault analysis software only after a configured number of corrected errors have been recorded.

Corrected error counters are described in the RAS System Architecture section in the Arm RAS for A-profile architecture supplement [10]. These support overflow detection, but not thresholding.

B.1.2.2 Interrupt enable controls

Architectural means to control the interrupts from error nodes are required. In particular, control is required to disable generation of interrupts on error correction, or to switch between polled and event-driven methods.

B.1.2.3 Standardized FRU and error location identification

The cost of going to firmware to decode FRU and location information can be high. This includes using an SMC or interpreter, for example an ACPI Machine Language (AML) [**ACPI?**] interpreter to do this on the

error recovery path. Some users do not want firmware to be able to mask errors from the kernel. Other system vendors have the requirement for firmware to mask the errors. Therefore, this requirement must be controllable. An ACPI Machine Language (AML) [**ACPI?**] or similar interpreter might not be available.

Note

An error location might be within a FRU, or might be within a Field Non-replaceable Unit (FNRU), for example within the SoC. A recorded error location is used for fault analysis.

B.1.2.4 Memory and cache ECC features

Memory, system caches, and large caches can have a long window of vulnerability for the data they store. Window of vulnerability is the period between accesses to the data, during which the data is liable to corruption. For a cache, a lower access rate for a location implies a larger window of vulnerability for that data.

Using Error Detection and Correction Code (EDAC) features is therefore important for the reliability of such memories.

If the data that is stored in a cache is always clean, then an error can be corrected by invalidating the copy in the cache. Otherwise, an error has to be correctable in-place.

Scrubbing reduces the window of vulnerability of data. This reduces the chance that single bit errors become multi-bit errors over time.

Poison is a technique that defers uncorrected errors to the point of their consumption. Poison provides the following advantages to a system:

- Poison provides a technique to contain latent errors. For example, when dirty data is evicted from a cache, the active process or virtual machine evicting the line is not consuming that data and might not even own the data. If an uncorrectable error is detected when dirty data is evicted from a cache, poison allows the error to be deferred to the real consumer of the data.
- Poison helps prevent false errors from generating failure. For example, a false error is generated if a failure occurs when an error is detected on a memory location that is not accessed, for example a location in a cache line that is not the location being accessed. If the poison granule is smaller than a cache line, poison can be allocated into a cache without causing software that does not access the error to fail.

  If the location is overwritten before it is accessed, no failure occurs. However, the error is recorded for fault analysis.

Poison therefore improves availability.

However, the details of EDAC features are IMPLEMENTATION DEFINED. This means that the details vary greatly depending on target customer markets and other design details. There is no one-size-fits-all solution to RAS features.

Therefore, SBSA has a mix of requirements and recommendations for base EDAC. Implementations are expected to go beyond these recommendations for certain markets.

B.1.3 Software faults

Historically it has been common to return an in-band error response to a software fault. (An in-band error response is the RAS term for an external abort.)

Examples of software faults include:

- Access to memory or device register that is not present. This includes cases where Secure and Non-secure memory are physically aliased.
- Access to a device that is not permitted at the device. For example, a Non-secure access to a Secure register.

- Access to a device that is in an inaccessible state or other illegal access. For example, the device is powered down, or the value written is not supported.

Some hardware faults are similar to software faults. For example, when a device surprise removal occurs, software is not at fault for attempting to access the now not-present device.

The Arm Architecture has no classification for External Abort exceptions that result from in-band error responses. This means that the historical approach leads to software faults being handled by the same software that processes hardware errors.

Also, there is no architecture to record syndrome for a software fault. This means that a conservative error handler might need to treat any software fault as a serious, potentially critical, error.

However, software faults are entirely predictable, usually containable, and avoidable. They should be treated as bugs.

Furthermore, if a device can be relied upon to return an in-band error response, that device must never be made available to any untrusted software, including a guest VM, if it could provide a denial-of-service attack vector.

The RAS System Architecture section in Arm RAS for A-profile architecture supplement [10] issue C.b made the recommended that errors classifiable as software faults generate software fault interrupts and do not return in-band error responses. However, this statement is replaced in issue D.a with the following recommendations:

- Where another standard defines a rule or sets a convention for a device, that should be followed. For example:
  - For a PCIe device, certain illegal accesses are RAO/WI.
  - [2] requires that *reserved accesses* to a component, such as reads and writes of unallocated or unimplemented registers and writes to read-only registers, behave as RAZ/WI.
  - [2] requires that under certain conditions accesses to certain debug registers return an error response.
- Accesses to a memory location that is not present can return an in-band error response when all of the following are true:
  - The location is not present due to a configuration of the physical address map that is either static or controlled by trusted software. For example, a configuration choice made by the designer, set during initial system configuration, or reconfigured by trusted software.
  - Within the aligned page that contains the not-present location, all other locations are also *not present* and have the same behavior. The size of this page is the largest supported translation granule size of all PEs in the system.

  That is, there is never any legitimate reason for software to access the page containing the location, and trusted software should set up the translation tables to prevent accesses from occurring.

For all other cases, the access should do one of the following:

- Return zeros for a read and ignore writes (RAZ/WI). This is the recommended behavior for reads and writes of unallocated or unimplemented registers, reads of write-only registers, and writes of read-only registers.
- Return all-ones for a read and ignore writes (RAO/WI).
- Return an IMPLEMENTATION DEFINED value for a read and ignore writes.

However, a device might implement a RAS System Architecture error node and error records for recording software faults, for improved debuggability of the fault.

When a device implements a RAS System Architecture error node and error records for recording software faults, software faults can be identified with specific error codes. Software faults are reported with an in-band error response or a *software fault interrupt* (a fault handling interrupt for software faults).

This should be configurable through ERR<n>CTLR, allowing software to disable the feature. For example, if an error exception might cause an unrecoverable software state.

When the feature is disabled, accesses should behave as recommended in the “all other cases” description above.

The following ERR<n>STATUS.SERR values can be used to record software faults:

SERR	Description
13	Illegal address (software fault). For example, access to unpopulated memory.
14	Illegal access (software fault). For example, byte write to word register.
15	Illegal state (software fault). For example, device not ready.
25	Error recorded by PCIe error logs. Indicates that the node has recorded an error in a PCIe error log. This might be the PCIe device status register, AER, DVSEC, or other mechanisms defined by PCIe.

If a device does not support a means to record the software fault, it should not return an in-band error response.

B.2 Server RAS architecture requirements

The following must be true of all PEs and other system components that implement the Arm RAS System Architecture [10] in the base server system.

R_{RAS_01} If the component is a memory controller or significant cache, it must implement one or more error counters following the standard programming model described in the Arm RAS System Architecture version 1.1 or later for at least all Corrected errors.

Within the standard programming model:

- The size of the error counter is an IMPLEMENTATION DEFINED choice of 8 or 16 bits (including an overflow bit).
- It is IMPLEMENTATION DEFINED whether a single counter, or a counter pair comprising a base counter and a repeat counter is implemented.

The choice of counter format should be based on the particular design parameters of the component.

Note

In this context, significant cache means any memory, cache, or buffer, where the expected corrected and uncorrected FIT rate is such that it might have a significant impact on product reliability or software.

The criteria by which a memory array is designated as a significant cache may depend on a range of design parameters, and therefore the designation is based on reliability requirements for the target market or customer.

Note

In this context, corrected errors does not necessarily mean single bit error. From the memory controller perspective, multi-bit errors (depending on the implementation) could still be considered “corrected”.

Also, memory controller corrected error counters (CEC) may only count errors that are detected and corrected at the controller.

For example, on systems that use DDR5 or any self-correcting memory technology, software needs to comprehend that the memory controller CEC may not provide the comprehensive count of all corrected DRAM errors.

Note

Future revisions of this specification may define additional RAS rules and recommendations for systems with DDR5 or other self-correcting memory technology.

R_{RAS_02} If the component is a memory controller or significant cache, it must follow the RAS System Architecture v1.1 or later as described in Arm RAS for A-profile architecture supplement [10]. The component must implement the CFI, DUI, and UI controls described in the RAS System Architecture v1.1 or later. That is, for the RAS node that records errors for the component, ERRFR.{CFI, DUI, UI} must each be either 0b10 or 0b11.

R_{RAS_03} Each error record group implements a single fault handling interrupt for all the records that are contained in the group.

R_{RAS_04} If any error record in an error record group is capable of generating an error recovery interrupt, the group implements a single error recovery interrupt for all the records contained in the group.

R_{RAS_05} If any error record in an error record group is capable of generating a critical error interrupt, the group implements a single critical error interrupt for all the records contained in the group.

R_{RAS_06} For each RAS interrupt that is generated by a RAS node that is accessible to a PE in the base server system:

- If the interrupt is a fault handling or error recovery interrupt then it must be connected to the base system GIC controller and is also permitted to be connected to a system component, for example a system control processor.
- If the interrupt is a critical error interrupt (CI), where the CI is routed to a system control processor, it is optional whether the CI is also routed to the GIC.

Unless otherwise required to be implemented as a PPI, the RAS interrupt is an SPI.

R_{RAS_07} If the component records a physical address (as required by the RAS System Architecture) for a fault at a location, and the address located in system memory is accessible to PEs in the base system, then the ERR<n>ADDR.AI bit, defined in Arm RAS for A-profile architecture supplement [10], must be 0b0 if the address is the same as System Physical Address for the location, and 0b1 otherwise.

I The System Physical Address (SPA) for a location is the physical address for the location that is used by PEs in the base server system.

I If the component records errors relating to more than one FRU, it should be possible for an operating system kernel fault handler to identify the FRU based on the information in the error record without requiring active firmware assistance.

I It should be possible for an operating system kernel fault handler to identify a fault location within a component based on the information in the error record without requiring active firmware assistance, for fault profiling and Predictive failure analysis. This includes locations in non-serviceable components.

R_{RAS_08} If an error record of the component is accessible through an *error record group*, then the ERRGSR (Error Group Status Register) as described in [10] reports the status of the error record.

B.2.1 Software faults

Examples of software faults include:

- Access to memory or device register that is not present. This includes cases where Secure and Non-secure memory are physically aliased.
- Access to a device that is not permitted at the device. For example, a Non-secure access to a Secure register.
- Access to a device that is in an inaccessible state or other illegal access. For example, the device is powered down, or the value written is not supported.

R_{RAS_10} Where the PCIe standard [1], Arm architecture [2], or other standard defines a rule or sets a convention for a software fault at a device, that rule or convention must be followed.

I For example:

- Arm ARM [2] requires that reserved accesses to a component, for example reads and writes of unallocated registers and writes to read-only registers, behave as RAZ/WI.
- [2] requires that under certain conditions accesses to certain debug registers return an error response.
- For a PCIe device, certain illegal accesses are RAO/WI.
- PCIe error rules are described in Section 1.6.7.3.

R_{RAS_11} On a software fault error if none of the following apply, a read returns an IMPLEMENTATION DEFINED value, and a write is ignored:

- The access is to a not present location.
- The response to the access is defined by RAS_10.

I It is recommended that the IMPLEMENTATION DEFINED value that is returned on a read of an unallocated, unimplemented, or write-only register is zero.

R_{RAS_12} For the purposes of the rule RAS_11, a location is defined as not present, only if all of the following apply:

- The location is not present due to a configuration of the physical address map that is either static or is controlled by trusted software.
  - A static configuration is a configuration that is made by the system designer, system integrator, or set during initial system configuration.
  - Controlled by Trusted software means that the location might be present or not present, but this is configured by Trusted software.
  - The split between trusted and untrusted is IMPLEMENTATION DEFINED. However, Untrusted would typically include unprivileged software and, in systems that supports virtualization, guest operating systems.
  - Untrusted might or might not include Non-secure hypervisors.
- Within the aligned page that contains the not-present location, all other locations are also not present and have the same behavior. The size of this page is the largest supported translation granule size of all PEs in the system.

I A device may also include a RAS error node to record an error due to software fault, to improve debugging of software faults. This node might also include controls to enable the return of an in-band error response, and a software fault interrupt, both of which should be disabled by default.

B.2.2 Recommended RAS features

This section is informative. This section describes RAS features which are recommended for all PEs and other system components in the base server system.

I It is recommended that each memory controller, system cache, and other large cache implements error detection.

Note

A large cache is one in which data might reasonably be expected to have a high window of vulnerability. It is recommended that Level 1 PE cache is classified as a large cache.

I It is recommended that caches that hold dirty data implement error correction with at least SECDED. It is also recommended that memory controllers protect external DRAM with higher level of ECC protection than that of the DRAM memory technology attached to it. For example: If the memory technology supports single error correction (SEC) on the DRAM memory, the memory controller is recommended to support at least one stronger ECC protection scheme on the data (for example, SECDED or Symbol ECC).

I It is recommended that each system cache and other large cache that can hold dirty data, and implements error detection, supports patrol scrubbing.

I It is recommended that each system cache and other large cache that holds only clean data and implements error detection does so to at least a SED standard. Detected errors must result in invalidation of the data.

C Self-hosted debug for Armv9-A

C.1 Goals

This chapter specifies hardware requirements for Armv9-A architecture revision and higher [2], where debug functionality running on an Operating System can rely on the hardware resources. It addresses PE features and key aspects of system architecture.

The primary goal is to ensure sufficient system architecture to enable a suitably-built driver and debug software framework to run on all hardware compliant with this section. It is anticipated that a machine-readable description of the hardware configuration is needed to ensure that the driver and debug software are appropriately configured for the specific system.

C.2 Levels of functionality

The requirements are introduced through levels of functionality, where each level provides a specified set of capabilities that software can rely on.

An implementation is consistent with a level of the Architecture if it implements all of the functionality of a given level, at performance levels appropriate for that particular level. This means that all of the functionality of a level can be exploited by software without unexpectedly poor performance.

While the levels are numbered, the numbering scheme does not always mean that software written for a particular level will work on hardware designed for any lower-numbered or higher-numbered level. For example, some higher-numbered levels restrict the options provided at lower-numbered levels. As such, software written only for the higher level might not work with hardware compliant with a lower-numbered level.

C.3 Self-hosted debug capabilities

A suitably-built debug software framework running on hardware compliant with one or more levels of this specification should be able to provide debug functionality consistent with the levels implemented. These capabilities include, but are not limited to:

- Tracing of PE program flow, providing detailed history of program execution. PE trace has multiple uses, including:
  - Post-mortem analysis.
  - Reverse debugging.
  - Performance analysis.
- Performance monitoring, providing detailed information about the performance and timing characteristics of programs running on a PE.

C.4 External debug capabilities

This chapter does not specify any capabilities for external debug. It is anticipated that external debug scenarios might use the same hardware functions as self-hosted debug scenarios however, and both external debug software and self-hosted debug software must accommodate the possibility that hardware functions might not be available because they are being used by another agent. To provide external debug functions, a system might include functionality or components in addition to those specified which might require initialization or programming to provide the self-hosted debug capabilities.

These requirements do not specify any mechanism for software agents to arbitrate over the use of hardware functions.

C.5 PE Trace

C.5.1 Background

I PE trace provides a detailed history of program flow, and is useful for both debugging and performance analysis. The objective of PE trace is to provide a debug software framework such as gdb or Linux perf, with a history of executed instructions, branches, or function calls.

I This specification details the following sets of requirements:

- The trace information in the generated trace and the observation capabilities of the trace functionality in each PE.
- The methods of capturing the generated trace.

C.5.2 Embedded Trace Extension (ETE)

I The levels of Embedded Trace Extension (ETE) provide the ability to generate a program trace and to set trace filtering trigger conditions.

C.5.3 ETE Level 1

I ETE Level 1 provides basic program flow tracing capability for all PEs in the system, with tracing provided from all PEs concurrently.

R_{ETE_01} In a system with multiple PEs, each PE that is to be used in the same operating system or hypervisor must be compliant with at least the same ETE level.

R_{ETE_02} Each PE in the system must be provided with a trace unit compliant with the Embedded Trace Extension (FEAT_ETE) [2].

R_{ETE_03} The trace unit must support the following ETE features:

- Cycle counting with a cycle counter that is at least 12-bits in size.
- At least one address comparator pair.
- At least one Context ID comparator.
- At least one Virtual context identifier comparator, if the PE implements EL2.
- At least one single-shot comparator control.
- At least one event in the trace.
- At least two counters.
- The sequencer state machine.
- At least four resource selection pairs.

I If a system provides control over the power to the trace unit, it is recommended that this control provides the means to conserve power when the trace unit is not used, including when transitioning between states where trace is used and not used.

R_{ETE_04} All trace units must share the same physical timestamp source.

R_{ETE_05} When TRFCR_EL1 and TRFCR_EL2 are used to select the time source, the selection of CoreSight time and Physical time must select the same time source. This means that there is no difference between CoreSight time and Physical time.

R_{ETE_06} The trace units for all PEs must be able to be enabled concurrently, and generate trace concurrently.

R_{ETE_07} Each PE in the system must implement the Trace Buffer Extension (FEAT_TRBE) [2].

R_{ETE_08} All the TRBE trace buffers must implement Flag Updates, or all TRBE trace buffers must not implement Flag Updates. TRBIDR_EL1.F must be the same for all TRBE trace buffers.

R_{ETE_09} All the TRBE trace buffers must implement the same minimum alignment constraints. TRBIDR_EL1.Align must be the same for all TRBE trace buffers.

R_{ETE_10} An implementation must reserve a PPI for the TRBE interrupt to PE.

I It is recommended that the TRBE and system interconnect infrastructure for capturing trace has sufficient bandwidth for common scenarios, while avoiding trace unit overflow scenarios. Common scenarios for tracing include:

- Continuous tracing of one PE, without cycle counts or branch broadcasting, with minimal impact on PE or system performance.
- Continuous tracing of all PEs concurrently, without cycle counts or branch broadcasting.
- Continuous tracing of one PE, with cycle counting and branch broadcasting enabled.

C.6 System Trace Macrocell

C.6.1 Background

I Arm defines a System Trace Macrocell (STM) Architecture [17] which enables tracing of instrumented software and system activity. The STM is used to generate trace for use either by external debuggers or self-hosted debuggers.

I Providing an STM implementation in a system provides a memory-mapped programmers model for software instrumentation libraries to target consistently across SoCs.

C.6.2 Level 1

The Trace Generation requirements in Section C.6.2.1 define a central STM unit for use by instrumentation software running on all PEs in a system.

The Trace Capture requirements in Section C.6.2.2 define self-hosted capture of the trace from an STM. This capture method ensures the PE trace and STM trace are captured independently, ensuring trace buffers for PE tracing can be independently managed.

The Trace Generation requirements defined in Section C.6.2.1 are identical to STG Level 1 defined in [9]. The Trace Capture requirements defined in Section C.6.2.2 are identical to STC Level 2 defined in [9].

C.6.2.1 Trace Generation (STG)

R_{STM_01} The system must provide an STM unit compliant with the STMv1.1 architecture as defined in [17].

R_{STM_02} The STM unit must implement the following options:

- STPv2 trace protocol. See [17].
- Absolute timestamping.
- At least one stimulus port master.
- Each master must implement at least 16384 Extended stimulus ports.
- A 64-bit fundamental data size.
- Invariant timing and guaranteed transaction types.
- The following Configuration Register options:
  - STMTSFREQR is read-write.
  - STMTSSTIMR is implemented.
  - STMSYNCR is implemented.
  - STMSPER is implemented.
  - STMSPTER is implemented.
  - Trigger control implements both multi-shot and single-shot.
  - STMSPOVERRIDER is implemented.
  - STMSPMOVERRIDER is implemented.
  - STMSPSCR is implemented.
  - STMSPMSCR is implemented.
  - STMTCSR.SYNCEN is always 0b1.
- Data compression on stimulus ports is either programmable or not implemented.

R_{STM_03} If any PE implements EL3, the STM unit must implement at least one stimulus port master which is only accessible by Secure software.

R_{STM_04} For systems with more than one PE, the system must use stimulus port masters according to one of the following:

- All PEs must be able to use the same stimulus port master.
- Each PE must be able to use a separate stimulus port master.

C.6.2.2 Trace Capture (STC)

I It is recommended that the system provides an independent trace buffer for the STM, based on shared system memory.

R_{STM_05} A trace buffer must be provided in the system to capture trace from the STM.

R_{STM_06} The STM must have a separate logical trace buffer based on shared system memory, based on one of the following:

- Shared system memory, based on a configuration of the CoreSight Embedded Trace Router (ETR), as defined in [18].
- Shared system memory, based on a configuration of the Embedded Trace Router (ETR) Architecture, as defined in [19].

R_{STM_07} The trace buffer must be able to be located anywhere in normal memory accessible by the PE.

R_{STM_08} The trace buffer must support the Circular Buffer mode of operation.

R_{STM_09} The trace buffer for the STM must not allow trace from any PE trace unit to be captured.

R_{STM_10} Where the trace buffer is based on shared system memory and supports a size greater than the smallest translation granule of the PEs, a page scattering mechanism must be provided to support the trace buffer being partitioned into separate physical pages each of which is no larger than the smallest translation granule of the system.

R_{STM_11} Where a page scattering mechanism is required, this must be based on one of the following:

- A System Memory Management Unit (SMMU) based on at least SMMUv1.
- The translation service provided by a CoreSight Address Translation Unit [20].

R_{STM_12} If the PEs in the system implement EL2, the page scattering mechanism must support two stages of translation using one of the following methods:

- Both stages are implemented using the CoreSight Address Translation Unit [20]. The CoreSight Address Translation Unit can provide both stages in a single step.
- Both stages are implemented using an SMMU.
- Stage 1 is implemented using the CoreSight Address Translation Unit, and stage 2 is implemented using an SMMU.

R_{STM_13} Where any part of the page scattering mechanism is implemented using an SMMU, the SMMU infrastructure must support independent streams for each trace buffer.

R_{STM_14} Where the trace buffer is based on shared system memory, the trace buffer must also support storage of trace into a contiguous physically-addressed buffer without a page scattering mechanism.

R_{STM_15} Where the trace buffer is based on shared system memory and the PE supports both Secure and Non-secure memory, the trace buffer must be able to be located in Non-secure memory, and it is IMPLEMENTATION DEFINED whether the trace buffer is able to be located in Secure memory.

R_{STM_16} Where a trace buffer can capture trace from multiple trace sources, the trace buffer must support packing all of the trace streams in the trace buffer using one of the following:

- The CoreSight formatting protocol, see [21].

R_{STM_17} Any trace buffer control component, page scattering mechanism, and any components that need to be programmed to ensure trace can reach the shared system memory must be accessible without the need for any system specific software other than that required to provide power to the component.

R_{STM_18} Any trace buffer control component, page scattering mechanism, and any components that need to be programmed to ensure trace can reach the trace buffer must be accessible directly in the Physical Address space of every PE.

I If a system provides control over the power to the trace buffer control components, page scattering mechanism, or any components that need to be programmed to ensure trace can reach the trace buffer memory, it is recommended that this control provides the means to conserve power when trace is not used, including when transitioning between states where trace is used and not used.

R_{STM_19} For all components that are involved in ensuring trace reaches the trace buffer, if the component implements the CoreSight authentication interface, system firmware must ensure the authentication interface is set to a condition which ensures trace will reach the trace buffer, before any Hypervisor or Operating System are started. If the PE implements EL3, only Non-secure debug is required to be enabled. If the PE does not implement EL3, debug must be enabled for the implemented security state. For more details on the authentication interface see [22].

R_{STM_20} Only the following programmable components must be involved in ensuring the trace reaches the trace buffer:

- CoreSight ATB Funnel.
- CoreSight ATB Replicator.
- CoreSight Trace Memory Controller, configured as an Embedded Trace FIFO (ETF).

R_{STM_21} For an STM with a dedicated output trigger signal, when this trigger signal is asserted, any trace buffer which captures trace from this STM must be able to detect the event has occurred, and must be able to use this event to stop trace capture. One of the following mechanisms must be used:

- Use of the trigger trace source ID value 0x7D as defined in [21], and use of the ATB Trigger Enable function in the STM.
- Use of a dedicated mechanism when a trigger occurs in the STM.

R_{STM_22} If the system uses the trace source ID value 0x7D, the trace buffer must support one of the following:

- Ignoring the trigger trace source ID other than for the purposes of using the event to control trace capture. The CoreSight ETR does not obey this rule.
- Packing of the trace streams such that the trigger source ID and its payload are embedded in the data stored in the trace buffer. The CoreSight ETR supports this operation.

R_{STM_23} Each trace buffer must be able to generate an interrupt to indicate when any of the following occur:

- In Circular Buffer mode, when the top of the trace buffer is reached.
- The trace buffer has reached a pre-programmed watermark fill level.

R_{STM_24} The trace buffer interrupt must be at least one of an SPI or LPI.

R_{STM_25} The trace buffer control component, page scattering mechanism, and any components that need to be programmed to ensure the trace can reach the trace buffer from the STM must not be reset on a Warm reset of the PEs.

R_{STM_26} The path from the STM to its trace buffer must not rely on any programmable components that also control the path for a different trace unit to its trace buffer.

R_{STM_27} The registers to control each trace buffer must be located in separate 64KB pages.

R_{STM_28} Any components which control the path of trace from the STM to the trace buffer and the page scattering mechanism must be located in separate 64KB pages from components which control the path of trace from a different trace unit.

R_{STM_29} Where the CoreSight Address Translation Unit is used for stage 2 translations, each CoreSight Address Translation Unit must be located in a separate 64KB page from all other trace control components.

I Where the CoreSight Address Translation Unit is used, it is recommended that the CoreSight Address Translation Unit is located in a separate 64KB page from all other trace control components.

D GPU accelerated compute

X This chapter lists a set of rules that must be met for correct and performant GPU accelerated compute. A set of recommendations are also given which when implemented would improve the performance of GPU accelerated compute.

R_{GPU_01} PCIe integration requirements as defined in Section 1.6.7.1 must be implemented.

R_{GPU_02} The system must support peer-to-peer read and write transactions between all Endpoints in the system, including those belonging to different Root Port downstream hierarchies. As described in the PCI Express Base Specification[1], there are two mechanisms by which a Root Complex may support peer-to-peer transactions:

1. The Root Complex "takes ownership" of the Transactions as a Completer and initiates separate Transactions as a Requester to the peer.
2. The Root Complex routes TLPs directly between peers. Note that in some configurations, Completions may take a different route than Requests and may not pass through the Root Complex.

R_{GPU_03} The peer-to-peer rules as defined in the Arm BSA [4] document are mandatory: PCI_PP_02, PCI_PP_03, PCI_PP_04, and PCI_PP_05. Additionally, the information statement PCI_PP_06 is a mandatory rule for all switches in the system.

R_{GPU_04} The Root Complex, Root Ports, and System MMUs in the system must support PCIe ATS. Root Complex and Root Port support for ATS must be indicated by firmware [3].

I System MMU support for ATS is indicated by the following SMMUv3 capability: SMMU_IDR0.ATS.

I It is recommended that the system supports atomic transactions, as advertized by the following Root Port capabilities:

- Device Capabilities 2 Register (Offset 24h): 32-bit AtomicOp Completer Supported
- Device Capabilities 2 Register (Offset 24h): 64-bit AtomicOp Completer Supported
- Device Capabilities 2 Register (Offset 24h): 128-bit CAS Completer Supported

I It is recommended that the system support peer-to-peer atomic transactions between all Endpoints in the system, extending GPU_02. This support is achieved in two parts:

1. Ensure each Root Port advertises the following capability: Device Capabilities 2 Register (Offset 24h): AtomicOp Routing Supported.
2. Ensure that there is no asymmetry of this capability and that AtomicOp Routing is supported between all Root Port pairs in the system. This strengthens the PCI Express specification, which allows for asymmetry.

I It is recommended that all NUMA nodes in the system have a symmetric PCIe topology with the same number of Root Port instances and symmetric lane width configurations for attaching accelerator and storage devices.

I Unless system address space is exhausted, each GPU device should be able to allocate the following amount of system address space:

- At least 16 MB for use as 32-bit non-prefetchable BAR memory region.
- At least 256 GB for use as 64-bit prefetchable BAR memory region.
- At least 32 MB for use as a second 64-bit prefetchable BAR memory region.

Some GPUs may require a legacy IO port space for legacy features like VGA. See Arm BSA [4] rule PCI_IO_01 for legacy IO port space requirement. The typical size is 128B.

I It is recommended that the amount of non-prefetchable memory space and prefetchable memory space allocated to the PCIe hierarchy below each Root Port is independently configurable on a per Root Port basis. The system should allow such configuration to be done post discovery of the downstream hierarchy's memory space needs.

Note

The intent of this recommendation is that a Root Port with a complex hierarchy (for example, including switches and multiple devices) can have more memory space allocated to its downstream hierarchy when compared to a Root Port with a single device directly attached to it.

I If there is no target for an address range in the system address map and that address range is above 4GB and below the maximum physical address limit, then mapping that address range to PCIe for use as 64 bit prefetchable BAR memory should be permitted.

I For performance reasons, it is recommended that:

1. Root Ports allow Inbound writes with RO=1 to overtake previously received Inbound writes provided the Root Port complies with the recommendation (2) below.
2. A write request with RO=1 is not permitted to overtake another write request if both writes are writing to bytes that fall within the same 64-bytes aligned granule in the system address space.

Note

64 bytes is the minimum granule size, performance of GPU compute usage models can benefit from larger granule sizes of up to 256 bytes.

I For performance reasons, it is recommended that a read is not permitted to overtake another read request if both reads are accessing bytes that fall within the same 64-bytes aligned granule in the system address space.

Note

64 bytes is the minimum granule size, performance of GPU compute usage models can benefit from larger granule sizes of up to 256 bytes.

I For performance reasons, it is recommended that the system is capable of supporting full saturation (for example, full utilization of the available link bandwidth) of the Root Port Inbound and Outbound links across a variety of traffic patterns. The following cases are provided for guidance:

1. Device to host memory (Inbound for Root Ports) read only, write only, and concurrent read and write traffic for all supported packet sizes greater than or equal to 128 bytes.
2. Device to device (peer-to-peer) read only, write only and concurrent read and write traffic for all supported packet sizes greater than or equal to 128 bytes.

Note

While verifying performance, a latency of 2.5us should be assumed between a non-posted request entering requester side Root Port pins to the last byte of the last completion for that request exiting requester side Root Port pins.

E CXL integration

The rules in this section apply to a base server system that supports CXL [5]. Such s system can support the CXL.mem protocol, the CXL.cache protocol or both.

R_{CXL_01} The version of the CXL specification [5] referenced for building an Arm-based CXL system must be CXL 2.0 or higher. This means that the host must not be an Exclusive Restricted CXL Host (eRCH) and must support the CXL VH mode by default.

I A VH-capable Downstream port in the host can operate in two modes: the CXL Virtual Hierarchy (VH) mode and the Restricted CXL Host Downstream port (RCH-DP) mode as defined in the CXL specification [5]. Support for RCH-DP mode is optional for a base server system but may be required for CXL compliance.

I If the host includes ports that support the RCH-DP mode, these ports must be capable of operating in the RCH-DP mode where an Exclusive Restricted CXL Device (eRCDs) is present in the hierarchy. When operating in this mode:

- The system will remap the port's config space to a memory-mapped RCRB block, as defined in the CXL specification [5]. The remapping is initiated when the CXL RCRB base address is programmed.
- he system will present an RCEC and associate the port with it. The RCEC must be capable of generating MSIs.

R_{CXL_02} An SMMU must be present in the CXL path if a Type 1 or Type 2 CXL device is supported. The SMMU must support CXL defined ATS messages.

R_{CXL_03} The system must present a memory-mapped region in the system address map that maps to the CHBCR register space.

R_{CXL_04} The system must implement the host-specific register/configuration spaces within the CHBCR, as specified in the CXL specification [5].

R_{CXL_05} Each CXL root port in the system must provide the means to send and receive VDMs mandated by the CXL specification [5].

R_{CXL_06} If a system implements firmware-first handling of memory error notifications, the host must provide a sink for incoming MEFN VDM messages.

R_{CXL_07} If a system supports MEFN feature for firmware-first reporting of CXL memory errors, a wired interrupt must be provided to notify the host of the occurrence of the memory error event.

I It is recommended that a base server system that supports CXL will support MPAM for CXL HDM, for the purpose of memory bandwidth control.

R_{CXL_08} The host implementation must support propagation of poison returned by a CXL device to the requesting PE, when the device detects an error while completing a demand read on CXL.mem. The PE must take a synchronous external abort on consumption of the poison.

R_{CXL_09} If a CXL Root port is unable to successfully complete an outbound CXL request, for example if Error Isolation is active or if the request is not supported or has timed out, then the port must support the following behavior:

- For CXL.mem requests that require a data response, return poison. For example, this applies to:
  - MemRd or MemRdData
- For CXL.mem requests that do not require a data response, return a non-data completion without signaling an in-band error. For example, this applies to:
  - MemWr or MemWrPtl
  - MemInv or MemClnEvct
- For CXL.cache H2D snoop requests the behavior must comply with the CXL specification. That is, if the host is tracking the device as a possible exclusive owner of the line, then data must be marked with poison.
- In all cases:
  - The CXL RP must complete the associated bus transactions in a compliant manner.

- The agent synthesizing the response is permitted to log the error details and generate an interrupt.

R_{CXL_10} If the system supports CXL-attached persistent memory, it must support PCMOs to that memory.

I Persistence guarantees for CXL-attached persistent memory can be provided by both host and device. A host might implement the following mechanisms in order to provide persistence guarantees:

- A hardware method for flushing all host caches and device caches to persistent memory whenever data must be made persistent.
- Issuing a CXL GPF (Global Persistence Flush) sequence to the device.
- Other host-specific means for persistence guarantees.

R_{CXL_11} The host implementation must provide means for guaranteeing that a PCMO has reached the Point of Persistence (PoP).

I For CXL devices, it is recommended that the host uses the write completion from the device as confirmation that a PCMO has reached the PoP of that device.

R_{CXL_12} The initial memory attributes associated with a CXL.cache transaction, before applying SMMU transformations, must be Normal Inner and Outer Write-back Cacheable Outer Shareable.

I Support for AArch64 architecture features that involve operations on system memory and thereby define the memory semantics on a system, must be presented consistently for all volatile system memory, including system memory provided by CXL Type-3 devices, regardless of whether these devices are present at system initialization or hot-added while the system is running.

I The following features are defined as *mandatory memory* features:

- Load-Exclusive/Store-Exclusive/Clear-Exclusive instructions
- FEAT_LSE (mandatory from Armv8.1), FEAT_LSE2 (mandatory from Armv8.4)

The following features are defined as *symmetric memory* features:

- FEAT_RME
- FEAT_MEC
- FEAT_RAS
- FEAT_MTE, FEAT_MTE2, FEAT_MTE_ASYNC
- FEAT_LS64, FEAT_LS64_V, FEAT_LS64_ACCDATA

This list is an exhaustive list of known features for existing architectural generations up to Arm v9.3.

Note

[14] specifies system requirements for enabling FEAT_RME and FEAT_MEC support for CXL memory.

R_{CXL_13} For CXL Type-3 memory that is to be presented to software as conventional memory:

- The host implementation of CXL must support all mandatory memory features.
- It is recommended that the host implementation will support any symmetric memory feature that is supported by the host for non-CXL memory.

Note

Conventional memory refers to memory locations from which generic OSs and application runtimes expect to create allocations for general software use.

I Global support for symmetric memory features is recommended since not supporting a certain memory feature for CXL can lead to that memory not being exposed to applications as conventional memory, due

to considerations related to CXL deployment in the system or to hypervisor and operating system software constraints.

I Host support for certain symmetric memory features may rely on device-side optional capabilities specified in CXL. For example, FEAT_MEC could rely on device-side encryption support. If software cannot establish that a CXL Type-3 device supports a capability that is required for implementing a symmetric memory feature then it can do the following:

- For memory that is present at OS boot, do one of the following:
  - Globally disable the feature for all conventional memory .
  - Not present the device as conventional memory.
- For memory that is discovered after OS boot:
  - Not present the device as conventional memory.

I Systems with CXL-attached memory might participate in memory pooling or sharing schemes involving CXL-attached memory. Such systems might require means for negotiating support of memory features with each other, and for disabling features that are not globally supported by all participant systems, which are outside the scope of this specification.