Arm® Server Base System Architecture 7.0

Platform Design Document

Non-confidential

Document number: DEN0029F

Contents

Release information

Version 7.0 (31 Jan 2021)

- Major additions are Level-7, Appendix B and C.
- Armv8.6 requirements (248).
- DMA masters 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).
- ERRADDR.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 6.0 (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 5.0 (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 standardisation (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 accesses 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 virtualised through FW (44).
- Clarify IO BAR usage (45).
- Generalise 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 3.0 (01 Feb 2016)

- Initial Release. Non-confidential.

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 PCIe specification [1] for more details.
AER	Advanced Error Reporting. A PCIe feature that enables software to isolate and analyze errors with fine granularity. See PCIe specification [1] for more details.
Arm ARM	Arm Architecture Reference Manual; see [2].
Arm recommends	This phrase is used in statements when Arm **suggests** that the implementer do something, but the decision remains with the implementer.
Arm strongly recommends	This phrase is used in statements when Arm **expects** that the implementer do something, but the decision remains with the implementer.
ATS	Address Translation Services
Base Server System	A system that is compliant with the Server Base System Architecture.
Completer	An agent in a computing system that responds to and completes a memory transaction that was initiated by a Requester.
ECAM	Enhanced Configuration Access Mechanism
GIC	Generic Interrupt Controller
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's cache.
LPI	Locality-specific Peripheral Interrupt (GICv3 [3]).
MMIO	Memory Mapped Input Output
P2P or Peer-to-peer	Traffic that is sent directly between two PCIe endpoints. See PCIe specification [1] for more details.
PCI Host Bridge (PHB)	A host-to-PCI bridge; this provides a design-time fixed range of bus and memory/IO resource ranges to the system. A PHB connects a set of root ports to the host.
PE	Processing Element, as defined in the Arm Architecture Reference Manual. Usually a single hardware thread of a PE.
PMU	Performance Monitoring Unit
PPI	Private Peripheral Interrupt
PRI	Page Request Interface
RCEC	Root Complex Event Collector
RCiEP	Root Complex integrated End Point
Requester	An agent in a computing system that is capable of initiating memory transactions.
Root Complex (RC)	A collection of PHBs and therefore RPs. A RC might be conceptually composed of one or more PCIe Host Bridge (PHB). A RC provides a PCI segment. All PHBs in a RC are on the same segment.

Term	Meaning
Root Port (RP)	A Root Port is associated with a physical port and appears in PCIe config space as a PCI bridge with a type 1 header, A RP is the root of a PCIe hierarchy.
SBBR	Server Base Boot Requirements [4].
SBSA	Server Base System Architecture.
SGI	Software-Generated Interrupt
SPI	Shared Peripheral Interrupt
SR-IOV	Single Root I/O virtualization. This is a method that allows a PCIe device to be virtualized. See PCIe specification [1] for more details.
System firmware data	System description data structures, for example ACPI or Flattened Device Tree.
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 5.0, version 1.0*. PCI-SIG.

[2] *DDI 0487 Arm® Architecture Reference Manual ARMv8, for the ARMv8-A architecture profile*. Arm Ltd.

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

[4] *DEN 0044 Arm Base Boot Requirements*. Arm Ltd.

[5] *DEN 0094A Arm® Base System Architecture version 1.0*. Arm Ltd.

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

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

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

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

[10] *NIST 800-90*. NIST. “https://csrc.nist.gov/publications/detail/sp/800-90b/final”.

[11] *SBSA ACS*. Arm Ltd. “https://github.com/ARM-software/sbsa-acs”.

[12] *CXL specification*. computeexpresslink.org.

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

[14] *Arm Architecture Reference Manual Supplement, The Scalable Vector Extension (SVE), for Armv8-A (ARMDDI 0584)*. Arm Ltd.

Rules-based writing

This specification consists of a set of individual rules. Each rule is clearly identified by the letter R.

Rules must not be read in isolation, and where more than one rule relating to a particular feature exists, individual rules are grouped into sections and subsections to provide the proper context. Where appropriate, these sections contain a short introduction to aid the reader. An implementation which is compliant with the architecture must conform to all of the rules in this specification.

Some architecture rules are accompanied by rationale statements which explain why the architecture was specified as it was. Rationale statements are identified by the letter X.

Some sections contain additional information and guidance that do not constitute rules. This information and guidance is provided purely as an aid to understanding the architecture. Information statements are clearly identified by the letter I.

Implementation notes are identified by the letter U.

Software usage descriptions are identified by the letter S.

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

Rules, rationale statements, information statements, implementation notes and software usage statements are collectively referred to as *content items*.

Identifiers

Each content item may have an associated identifier which is unique within the context of this specification.

When the document is prior to beta status:

- Content items are assigned numerical identifiers, in ascending order through the document (*0001*, *0002*, . . .).
- Identifiers are volatile: the identifier for a given content item may change between versions of the document.

After the document reaches beta status:

- Content items are assigned random alphabetical identifiers (*HJQS*, *PZWL*, . . .).
- Identifiers are preserved: a given content item has the same identifier across versions of the document.

Examples

Below are examples showing the appearance of each type of content item.

R This is a rule statement.

R_{X001} This is a rule statement.

I This is an information statement.

X This is a rationale statement.

U This is an implementation note.

S This is a software usage description.

Feedback

Arm welcomes feedback on its documentation.

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

- The title (Server Base System Architecture).
- The document ID and version (DEN0029F 7.0).
- 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

1.1 Arm Base System Architecture

This document is a supplement to the Arm Base system Architecture document version 1.0 [5].

1.2 Introduction

Server Base System Architecture (SBSA) specifies a hardware system architecture, based on Arm 64-bit architecture, that server system software, for example operating systems, hypervisors, and firmware can rely on. SBSA extends the requirements specified in the Arm BSA [5].

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

All views of a level are required to be implemented to be compliant to a level.

SBSA Compliance is a requirement for the SR and LS bands of the Arm SystemReady program.

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 can be exploited by software without unexpectedly poor performance.

The SBBR recipe in the Arm Base Boot Requirements (Arm BBR) specification [4] describes firmware requirements for an Arm server system. Where this specification refers to system firmware data, it refers to firmware specified in the Arm BBR.

1.2.1 SBSA levels and minimum component versions

Table 2 summarizes the minimum architecture versions that map to the rules that are mentioned in a 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.

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	v3.2	v3.0
7	v8.6	v3.2	v3.0

Note

Each SBSA levels 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 backported even further. Please refer to extension documents for more details. Table 2 indicates the version of the architecture extension at which the required features were introduced.

Note

The levels between Level 3 and Level 6 that are presented in this document are semantically equivalent to the corresponding levels in SBSA 6.0 [6].

1.3 Level 3

$R_{S\_L3\_01}$ The rules from Arm BSA, [5] as listed in Section 1.8.1 must be implemented.

I The PE Security requirements in Arm BSA [5] are relaxed to become a recommendation. This is to maintain consistency with systems which are certified as ServerReady v1.0.

1.3.1 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 granule.

$R_{S\_L3PE\_04}$ Server base systems that make use of FEAT_LPA must provide a mode where the memory map is wholly contained inside 2^{48} bytes (256TB). This is required to support operating systems that do not use the 64KB granule.

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

1.3.2 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.

1.3.3 Interrupt controller

$R_{S\_L3GI\_01}$ A base server system must implement an interrupt controller that is compliant with the GICv3 or higher architecture [3].

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

1.3.4 PPI assignments

$R_{S\_L3PP\_01}$ The Interrupt IDs must be the same as the recommended values mentioned in Arm BSA [5].

1.3.5 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].

1.3.6 Watchdog

$R_{S\_L3WD\_01}$ The server base system must implement a Non-secure Generic watchdog as described in Watchdogs section in Arm BSA [5].

1.4 Level 4

The list of rules to be implemented for Level 4 is available in Section 1.8.2.

1.4.1 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.

1.4.2 System MMU and device assignment

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

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

$R_{S\_L4SM\_03}$ The integration of the System MMUs is compliant with the rules in SMMUv3 integration appendix in Arm BSA [5].

1.4.3 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 [7].

1.5 Level 5

The list of rules to be implemented for level 5 is available in Section 1.8.3.

1.5.1 PE architecture

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

RS_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. Arm recommends Level 2. See Section 1.5.3 for the equivalent requirements for the SMMU.

RS_L5PE_02 All PEs must implement address authentication using the standard algorithm that is defined by the Arm architecture [2], as indicated by ID_AA64ISAR1_EL1.APA. See FEAT_PAuth in [2].

RS_L5PE_03 PEs that are based on Armv8.4 must implement the requirements of the CS-BSA combination C [8].

I CS-BSA combination C[8] requires that PEs based on Armv8.4 implement the Armv8.4 Self-hosted trace extension. See FEAT_TRF in [2].

RS_L5PE_04 All PEs must implement the Activity Monitors Extension (AMU).

RS_L5PE_05 Where export control allows, all PEs must implement cryptography support for SHA3 and SHA512. See FEAT_SHA3 and FEAT_SHA512 in [2].

Note

Arm recommends that FEAT_SM3 and FEAT_SM4 is also supported in hardware aimed at the Chinese market.

RS_MPAM_PE Implementation of the MPAM extension (FEAT_MPAM) is OPTIONAL, however if implemented, the following minimal requirements must be met by the implementation:

- provide a minimum of 16 physical partition IDs.
- provide virtualization support with a minimal of 8 virtual partition IDs.
- provide a minimal of 2 performance monitor groups.
- provide cache portion partitioning in the last level cache.

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

Note

Arm strongly recommends that the number of partition IDs scales with the number of PEs.

RS_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].

RS_L5PE_07 All PEs must implement enhanced nested virtualization, that is provided by HCR_EL2.NV2 and the VNCR_EL2 register. See FEAT_NV2 in [2].

1.5.2 Interrupt controller

RS_L5GI_01 Only a GIC v3, or higher, interrupt controller will be visible to operating system software. Other forms of interrupt controller, for example interrupt combining or forwarding engines, are not permissible, if they require a platform specific kernel driver.

1.5.3 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.

Note

Arm recommends the SMMU implements Level 2. If the SMMU implementation provides Level 2, then Arm recommends that the PE also provides level 2.

X MPAM architecture requires that all requesters that can access an MPAM controlled resource, must support passing MPAM ID information.

$R_{S\_L5SM\_03}$ An SMMUv3.2 implementation must support the MPAM extension if the requests it serves access MPAM controlled resources.

1.5.4 Clock and Timer subsystem

1.5.4.1 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.

Note

Arm strongly recommends that this type of system uses revision 1 of the generic watchdog. This is because at 1Ghz the watchdog timeout refresh period is limited to just over 4s.

1.5.5 Watchdog

I Arm strongly recommends 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.

1.5.6 PPI assignments

$R_{S\_L5PP\_01}$ In addition to the PPI assignment specified in Arm BSA [5], 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.
1088-1103	Reserved	Reserved for future SBSA usage.

1.6 Level 6

The list of rules to be implemented for Level 6 is available in Section 1.8.4.

1.6.1 PE architecture

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

$R_{S\_L6PE\_01}$ PE security requirements as described in [5] are mandatory.

$R_{S\_L6PE\_02}$ 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 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

Arm strongly recommends that, the server base 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].

1.6.2 System MMU and device assignment

$R_{S\_L6SM\_01}$ The SMMU must implement coherent access to in memory structures, queues, and page tables as indicated by SMMU_IDR0.COHACC = 0b1.

$R_{S\_L6SM\_02}$ 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.

1.6.3 Watchdog

$R_{S\_L6WD\_01}$ The generic watchdog revision must be v1, W_IIDR==0001b.

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

1.6.4 Armv8 RAS extension requirements

$R_{S\_RAS\_01}$ PEs and other system components that implement the Armv8 RAS extension [9] must:

- Use Private Peripheral Interrupts for ERI or FHI if the only interface available for a RAS node is System register based.

- Use the generic counter time base if the timestamp extension is implemented. This means that ERR<n>FR.TS must be either b00 or b01.

$R_{S\_RAS\_02}$ Support for error injection is OPTIONAL, however Arm recommends 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.

I Arm recommends that the Timestamp error extension is implemented.

1.7 Level 7

The list of rules to be implemented for Level 7 is available in Section 1.8.5.

1.7.1 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].

$R_{S\_L7PE\_03}$ PEs must implement the enhancements to the Activity Monitor Unit (AMU) as indicated by ID_AA64PFR0_EL1.AMU == b0010. See FEAT_AMUv1p1 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 EnhancedPAC2 and FPAC extensions as indicated by ID_AA64ISAR1_EL1.APA == b0101. See FEAT_PAuth2 in [2].

$R_{S\_L7PE\_07}$ 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].

$R_{S\_L7PE\_08}$ Arm recommends the implementation of the data gathering hint feature as indicated by ID_AA64ISAR1_EL1.DGH == b001. See FEAT_DGH in [2].

$R_{S\_L7PE\_09}$ Arm recommends the implementation of the WFE fine tuning delay feature as indicated by ID_AA64MMFR0_EL1. TWED == b0001. See FEAT_TWED in [2].

$R_{S\_L7PE\_10}$ Arm strongly recommends that enhanced PAN feature is implemented, which is introduced inArm v8.7-A but can be implemented from v8.1-A. Support for enhanced PAN is indicated by ID_AA64MMFR1_EL1.PAN == b0011.

1.7.1.1 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 loads 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.

1.7.1.2 *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, committing and transaction must not be higher than the latency of the code sequence that Arm recommends 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.

$R_{S\_L7TME\_4}$ Arm recommends using the hardware cache coherency facilities of the processor to detect transactional conflicts. This is also known as eager conflict detection.

$R_{S\_L7TME\_5}$ Arm recommends 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.

1.7.2 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.

$R_{S\_L7MP\_03}$ The implementation must provide MPAM Cache Storage Usage monitors for the last-level 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.

$R_{S\_L7MP\_06}$ The MBWUs must implement the MPAM v1.1 64-bit MBWU extension. See FEAT_MPAMv1p1 in [2].

$R_{S\_L7MP\_07}$ The MBWUs for an interface must be sized so that they can count the total number of bytes that are transferred in a ten-second window when operating at maximum capacity.

X The 64-bit MBWU extension allows counters to be either 44 bits or 63 bits. The effect of the above constraint is that:

- A 44-bit counter is sufficient for a single interface of up-to 1.6TB/s.
- A 63-bit counter is necessary otherwise.

I Arm recommends that the number of MBWU scale with the number of PEs in the system.

$R_{S\_L7MP\_08}$ 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.

1.7.3 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 [10], 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 [11].

1.7.4 SMMU and device assignment

I If an SMMU is in the path of a CXL device [12], 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.

$R_{S\_L7SM\_01}$ All DMA capable requesters that are visible to the normal world PE software must be behind an 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 Arm recommends that an ETR [8] 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 [8] 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.

1.7.5 Performance Monitoring Unit

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

1.7.6 System RAS

R_{S_L7RAS} The rules in Section C of this specification must be implemented.

The following must be true of all PEs and other system components in the base server system.

1.7.6.1 Scrubbing

$R_{L7\_RAS\_01}$ 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.

Arm recommends that control features and IMPLEMENTATION DEFINED capabilities are accessible to an operating system through a common interface, for example ACPI RASF. ACPI RASF supports three patrol speed levels: slow, medium, and fast.

1.7.6.2 Poison

$R_{L7\_RAS\_02}$ The system must support the storage and forwarding of poisoned values.

I Arm recommends that poison is generated when an uncorrectable 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 granule. The poison is propagated and expanded if required.

Note: Generating poison can *defer* the error.

$R_{L7\_RAS\_03}$ When an uncorrectable error is detected, a component within the system may not be able to generate poison, propogate poison, or write poison. In this scenario, the component must generate one or more in-band error response, for example an External Abort. If enabled, an error recovery interrupt, as described by [9] must also be generated.

For example, when:

- An uncorrectable error is detected in data received by a device and the device cannot defer the error.
- An uncorrectable error is detected retrieving from a transaction queue within a memory controller, and the memory does not support poison.
- Poisoned data is received by a component that does not support poison, or has support disabled.

- Corruption is detected in data being read from memory, but the consumer cannot accept poison. For example, the data is being read by a processing engine that requires valid data, for example, a CPU pipeline.

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.

Arm recommends that if removing poison is supported, systems provide details of the method or a means to execute it in publicly available documentation.

1.7.7 PCIe

In addition to the rules in Arm BSA [5] for PCIe root ports and endpoints, the following rules must be implemented.

1.7.7.1 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.

1.7.7.2 i-EP

X This rule 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 4.2.6.9 in [1].

1.7.7.3 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].

$R_{PCI\_ER\_02}$ The Root Port must implement the MSI capability.

$R_{PCI\_ER\_03}$ The Root Port must report reception of PCIe error messages using MSI interrupts. See section 6.2.6 and 6.2.4.2 of [1].

I The Root Port error reort 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 Reference [1].

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

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.

All rule IDs which begin with ‘B’ are defined in the Arm BSA document [5].

1.8.1 SBSA Level 3 checklist

PE	Rule ID
Operating system	
	B_PE_01
	B_PE_02
	B_PE_03
	B_PE_04
	B_PE_05
	B_PE_06
	B_PE_07
	B_PE_08
	B_PE_09
	B_PE_10
	B_PE_11
	B_PE_12
	B_PE_13
	B_PE_14
	S_L3PE_01
	S_L3PE_02

PE	Rule ID
	S_L3PE_03
	S_L3PE_04
Hypervisor	
	B_PE_18
	B_PE_19
	B_PE_20
	B_PE_21
	B_PE_22
Platform security	
	B_PE_23
	B_PE_24

Memory map	Rule ID
Operating system	
	B_MEM_01
	B_MEM_02
	B_MEM_03
	B_MEM_04
	B_MEM_05
	B_MEM_06
	B_MEM_07
	S_L3MM_01
	S_L3MM_02
Platform security	
	B_MEM_08
	B_MEM_09

Interrupts	Rule ID
Operating system	
	S_L3GI_01
	S_L3GI_02
	B_GIC_03
	B_GIC_04

Interrupts	Rule ID
	B_GIC_05
Operating system	
	S_L3PP_01
	B_PPI_01
Hypervisor	
	B_PPI_02
Platform security	
	B_PPI_03

SMMU	Rule ID
Operating system	
	B_SMMU_01
	B_SMMU_02
	B_SMMU_06
	B_SMMU_07
	B_SMMU_08
	B_SMMU_12
	S_L3SM_01
Hypervisor	
	B_SMMU_16
	B_SMMU_17
	B_SMMU_18
	B_SMMU_19
	B_SMMU_21

Timer subsystem	Rule ID
Operating system	
	B_TIME_01
	B_TIME_02
	B_TIME_03
	B_TIME_04
	B_TIME_05
	B_TIME_06

Timer subsystem	Rule ID
	B_TIME_07
	B_TIME_08
	B_TIME_09
	B_TIME_10

Power and wakeup	Rule ID
	B_WAK_01
	B_WAK_02
	B_WAK_03
	B_WAK_04
	B_WAK_05
	B_WAK_06
	B_WAK_07
	B_WAK_08
	B_WAK_09
	B_WAK_10
	B_WAK_11

Peripherals	Rule ID
Operating system	
	B_PER_01
	B_PER_02
	B_PER_03
	B_PER_04
	B_PER_05
	B_PER_06
	B_PER_07
	B_PER_08
	B_PER_09
	B_PER_10
	B_PER_12
Platform security	
	B_PER_11

Watchdog	Rule ID
	B_WD_01
	B_WD_02
	B_WD_03
	B_WD_04
	B_WD_05
	B_WD_06

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	~~B_SMMU_08~~
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
PE	S_MPAM_PE

Module	Rule ID
GIC	S_L5GI_01
SMMU	~~S_L4SM_01~~
SMMU	S_L5SM_01
SMMU	S_L5SM_02
SMMU	S_L5SM_03
SMMU	S_L5SM_04
SMMU	B_SMMU_09
SMMU	B_SMMU_11
SMMU	B_SMMU_20
SMMU	B_SMMU_22
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	B_PE_16
PE	B_PE_17
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	B_SEC_01
PE	B_SEC_02
PE	B_SEC_03
PE	B_SEC_04
PE	B_SEC_05
SMMU	B_SMMU_03
SMMU	B_SMMU_04
SMMU	B_SMMU_05
SMMU	B_SMMU_13
SMMU	B_SMMU_14

Module	Rule ID
SMMU	B_SMMU_23
SMMU	S_L6SM_01
SMMU	S_L6SM_02
SMMU	S_L6SM_03
Watchdog	S_L6WD_01
RAS	S_RAS_01
PCI	B_REP_1
PCI	B_IEP_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_03
PE	S_L7PE_04
PE	S_L7PE_05
PE	S_L7PE_06
PE	S_L7PE_07
PE	S_L7PE_08
PE	S_L7PE_09
PE	S_L7PE_10
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
PE MPAM	S_L7MP_03
PE MPAM	S_L7MP_04
PE MPAM	S_L7MP_05

Module	Rule ID
PE MPAM	S_L7MP_06
PE MPAM	S_L7MP_07
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	S_L7RAS
RAS	S_L7RAS_1
RAS	S_L7RAS_2
RAS	S_L7RAS_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_02
PCIe	PCI_ER_03
PCIe	PCI_ER_04
PCIe	PCI_ER_05
PCIe	PCI_ER_06

A Level 3 - firmware

Level 3 - firmware is an optional additional set of requirements for a Level 3 system. Level 3 - firmware is designed to give a base set of functionality that standard platform firmware can rely on. A system that is compliant with level 3 and not compliant with level 3 - firmware is still a fully compliant level 3 system. The system has all the features required by the operating systems and hypervisors.

A.1 Memory map

The system must provide some memory mapped in the Secure address space. The memory must not be aliased in the Non-secure address space. The amount of Secure memory provided is platform-specific as the intended use of the memory is for platform-specific firmware.

All Non-secure on-chip masters in a base server system that are expected to be used by the platform firmware must be capable of addressing all of the Non-secure address space. If the master goes through a SMMU then the master must be capable of addressing all of the Non-secure address space even when the SMMU is off.

A.2 Clock and Timer Subsystem

A system compatible with level 3 - firmware must also include a Secure wakeup timer in the form of the memory mapped timer described in the Armv8 ARM [2] This timer must be mapped into the Secure address space, and the timer expiry interrupt must be presented to the GIC as an SPI. This timer is called the Secure system wakeup timer.

The Secure wakeup timer does not require a virtual timer to be implemented. The virtual offset register can Read-As-Zero, where writes to the virtual offset register in CNTCTLBase frame are ignored. The timer is not required to have a CNTEL0Base frame.

The following table summarizes which address space the register frames related to the Secure wakeup timer should be mapped on to.

Register Frame	
CNTControlBase	Secure
CNTReadBase	Not required
CNTCTLBase	Secure
CNTBaseN	Secure

CNTCTLBase might be shared among multiple timers, including various Secure and Non-secure timers. The SBSA specification does not require this.

Note

- GICv3 does not support Secure LPI. Therefore, the Secure system timer interrupt must not be delivered as LPI.
- It is recognized that in a large system, a shared resource like the system wakeup timer can create a

system bottleneck. This is because, access to it must be arbitrated through a system-wide lock. Level 3-firmware requires just a single timer so that standard firmware implementations have a guaranteed timer resource across platforms. We anticipate that large PE systems will implement a more scalable solution like one timer for each PE.

A.3 Watchdog

The required behavior of watchdog signal 1 of the Non-secure watchdog is modified in level 3 - firmware and is required to be routed as an SPI to the GIC. It is expected that this SPI be configured as an EL3 interrupt, directly targeting a single PE.

A system compatible with level 3- firmware must implement a second watchdog, and is referred to as the Secure watchdog. It must have both its register frames mapped in the Secure memory address space and must not be aliased to the Non-secure address space.

Watchdog Signal 0 of the Secure watchdog must be routed as an SPI to the GIC and it is expected this will be configured as an EL3 interrupt, directly targeting a single PE.

Note

- GICv3 does not support Secure LPI. The Secure watchdog interrupts must not be delivered as LPI.
- Only directly-targeted SPIs are required to wake a PE. Programming the watchdog SPI to be directly targeted ensures delivery of the interrupt independent of PE power states. However, it is possible to use a 1 of N SPI to deliver the interrupt, if one of the target PEs is running. See Arm BSA [5] for information about SPI waking a PE.

Watchdog Signal 1 of the Secure watchdog must be routed to the platform. In this context, platform means any entity that is more privileged than the code running at EL3. Examples of the *platform* component that services Watchdog Signal 1 are a system control processor, or dedicated reset control hardware.

The action that is taken on the raising of Watchdog Signal 1 of the Secure watchdog is platform specific.

B 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.

B.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.

B.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. Arm recommends that the INST_RETIRED event is implemented.

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

- 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 documention 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 Arm recommends 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.
- Arm recommends that applicable implementation details are provided in freely-usable, machine-readable standard formats.

B.3 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
0x0008	INST_RETIRED	Instruction architecturally executed
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
0x0032	LL_CACHE	Last Level data cache access
0x0033	LL_CACHE_MISS	Last level data cache miss
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:

- MEM_ACCESS
  - L1D_TLB
    - DTLB_WALK
- >L1I_TLB
  - ITLB_WALK

$R_{PMU\_EV\_03}$ The events that are listed in the following table measure TLB effectiveness, for each applicable level of PE TLB:

Number	Mnemonic	Description
0x0013	MEM_ACCESS	Data memory access
0x0025	L1D_TLB	Level 1 data TLB access

Number	Mnemonic	Description
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

$\mathrm{R}_{\mathrm{PMU\_EV\_04}}$ 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.

X The following events are designed for accounting for cycles.

- CPU_CYCLES
  - STALL
    - STALL_FRONTEND
  - STALL
    - STALL_BACKEND
      - STALL_BACKEND_MEM
  - Not stalling

$\mathrm{R}_{\mathrm{PMU\_EV\_05}}$ 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
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

X This is the rationale for the rule PMU_EV_06.

B.3.1 Fractional cycle accounting

The following events are designed for accounting for fractional cycles:

- CPU_CYCLES SLOTS
  - STALL_SLOT
    - STALL_SLOT_FRONTEND
  - STALL_SLOT
    - STALL_SLOT_BACKEND
  - *Not stalling*

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.

B.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:

- OP_SPEC
  - OP_RETIRED

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

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

B.3.3 Top-down accounting

Top-down accounting combines the Section B.3.1 and Section B.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$_{\text{PMU\_EV\_06}}$ The events in the following table must be implemented for Section B.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

X This is the rationale for the rule PMU_EV_07.

B.3.4 Workload events (SVE)

SVE [14] defines pairs of events for measuring vector and scalar operation workloads.

- FP_FIXED_OPS_SPEC
  - FP_HP_FIXED_OPS_SPEC
  - FP_SP_FIXED_OPS_SPEC
  - FP_DP_FIXED_OPS_SPEC
- INT_FIXED_OPS_SPEC
- FP_SCALE_OPS_SPEC
  - FP_HP_SCALE_OPS_SPEC
  - FP_SP_SCALE_OPS_SPEC
  - FP_DP_SCALE_OPS_SPEC
- INT_SCALE_OPS_SPEC

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 as multiply-accumulate and dot products, count as multiple operations.

Although these events are defined by [14], the FIXED events can be implemented on a PE that does not include [14].

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
0x80C9	INT_FIXED_OPS_SPEC	Non-scalable integer 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
0x80C8	INT_SCALE_OPS_SPEC	Scalable integer element operations speculatively executed

X This is the rationale for the rule PMU_EV_08.

B.3.5 Branch predictor effectiveness

The following events are designed for monitoring branch predictor effectiveness.

- BR_RETIRED
  - BR_PRED_RETIRED
    - BR_IMMED_PRED_RETIRED
    - BR_IND_PRED_RETIRED
      - BR_INDNR_PRED_RETIRED
      - BR_RETURN_PRED_RETIRED
  - BR_PRED_RETIRED
    - BR_TAKEN_PRED_RETIRED
    - BR_SKIP_PRED_RETIRED
  - BR_MIS_PRED_RETIRED
    - BR_IMMED_MIS_PRED_RETIRED
    - BR_IND_MIS_PRED_RETIRED
      - BR_INDNR_MIS_PRED_RETIRED
      - BR_RETURN_MIS_PRED_RETIRED
  - BR_MIS_PRED_RETIRED
    - BR_TAKEN_MIS_PRED_RETIRED
    - BR_SKIP_MIS_PRED_RETIRED

$R_{PMU\_EV\_08}$

The events listed in the following table are use to measure Section B.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
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.

X This is the rationale for the rule PMU_EV_10.

B.3.6 Latency

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.

The events listed in the table below are recommended for latency measurements:

Mnemonic	**Description**
INST_FETCH	Instruction memory access.
INST_FETCH_CYCLES	Total cycles, INST_FETCH.
MEM_ACCESS_CYCLES	Total cycles, MEM_ACCESS.
MEM_ACCESS_RD_CYCLES	Total cycles, MEM_ACCESS_RD
MEM_ACCESS_WR_CYCLES	Total cycles, MEM_ACCESS_WR
BUS_ACCESS_CYCLES	Total cycles, BUS_ACCESS
BUS_ACCESS_RD_CYCLES	Total cycles, BUS_ACCESS_RD
BUS_ACCESS_WR_CYCLES	Total cycles, BUS_ACCESS_WR
DTLB_WALK_CYCLES	Total cycles, DTLB_WALK
ITLB_WALK_CYCLES	Total cycles,ITLB_WALK

$R_{PMU\_EV\_10}$ The events listed in the following table are used to measure Section B.3.6:

Number	Mnemonic	Description
0x0013	MEM_ACCESS	Data memory access
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

$R_{PMU\_EV\_11}$ 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.

B.3.7 Memory workload

X The rationale for memory workload events is as follows.

The SVE [14] feature defines a pair of vector and scalar memory workload events.

- LDST_FIXED_OPS_SPEC
  - LD_FIXED_OPS_SPEC
  - ST_FIXED_OPS_SPEC
- LDST_SCALE_OPS_SPEC
  - LD_SCALE_OPS_SPEC
  - ST_SCALE_OPS_SPEC
- LDST_FIXED_BYTES_SPEC
  - LD_FIXED_BYTES_SPEC
  - ST_FIXED_BYTES_SPEC
- LDST_SCALE_BYTES_SPEC
  - LD_SCALE_BYTES_SPEC
  - ST_SCALE_BYTES_SPEC

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 [14], 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 Arm recommends that the events listed in the table below are implemented for measuring Section B.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

Number	Mnemonic	Description
0x80DF	ST_FIXED_BYTES_SPEC	Non-scalable store bytes speculatively executed

If SVE is implemented, Arm recommends the events listed in the following table are implemented for for measuring Section B.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 Arm recommends 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

If SVE is implemented, Arm recommends 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

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

B.3.8 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 A unidirectional read/write interface supports reads and writes that originate on one side of the interface. A memory interface is an example of a unidirectional interface.

I A bidirectional read/write interface supports read or write requests originating on either side of the interface. If the monitor is considered to be on one side of this interface, then these 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.

The monitor 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 interface.

I A unidirectional streaming interface supports writes originating on one side of the interface.

I A bidirectional streaming interface supports writes originating on either side of the 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 Arm recommends that measurements that are generated by event monitors are implemented in a way that is compatible with the *Arm CoreSight Performance Monitoring Unit Architecture*.

Note: Arm CoreSight-PMU specification is not yet publicly available.

The base server system must implement bandwidth monitors for:

- $R_{PMU\_BM\_1}$ Memory interface.
- $R_{PMU\_BM\_2}$ PCIe interface.
- $R_{PMU\_BM\_3}$ External accelerator interface.
- $R_{PMU\_BM\_4}$ 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 a unidirectional read/write interface must be capable of measuring all of any of the following groups of measurements simultaneously:

- Total bandwidth, and average read latency
- Read bandwidth, write bandwidth, and average read latency

$R_{PMU\_SYS\_2}$ Each monitor for a bidirectional read/write interface must be capable of measuring all of any of the following groups of measurements simultaneously:

- Total outbound bandwidth, outbound average read latency, total inbound bandwidth, and inbound average read latency.
- Outbound read bandwidth, outbound write bandwidth, outbound average read latency, inbound read bandwidth, inbound write bandwidth, and inbound average read latency.

$R_{PMU\_SYS\_3}$ Each monitor for a unidirectional streaming interface must be capable of measuring total bandwidth.

$R_{PMU\_SYS\_4}$ Each monitor for a bidirectional streaming interface must be capable of measuring total inbound bandwidth and total outbound bandwidth 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, each monitor must be 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.

Mnemonic	Description
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.

B.3.8.1 Recommendations

Arm recommends that the base server system implement the monitors that are specified in this section.

I Arm recommends that the base server system implement average latency monitors for each:

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

I Arm recommends that the base server system implement bandwidth monitors for each:

- External PCIe device
- High-bandwidth internal device

I Arm recommends 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.

B.3.8.2 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.

C Server RAS

The Arm RAS Extension and Arm RAS System Architecture 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.

C.1 Justifications and impact

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

For Arm server systems, Arm recommends setting a higher standard of features, 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.

C.1.1 PE architecture

C.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 manner that allows software to contain and possibly recover from an error.

C.1.2 System architecture components

This section includes components of a CPU that the RAS System Architecture treats as system components, for example caches.

C.1.2.1 Error counters

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

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. These support overflow detection, but not thresholding.

C.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.

C.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 ASL 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 ASL or similar interpreter might not be available.

Note

The RAS supplement [9] issue C.b used FRU to mean both:

- A Field Replaceable Unit, for example a DIMM
- An error location

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.

This is corrected in issue D.a.

C.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 interrupts the window of vulnerability of data. This reduces the chance that single bit errors become multi-bit errors.

Poison is a technique that defers 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 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 a quality of implementation issue. 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.

C.1.3 Software faults

Historically it has been common to return an in-band error response in 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 such 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 RAS faults. In addition, 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 [9] 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. 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.
- 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 recorded as an error. Faults are reported with an in-band error response or a fault handling interrupt (software fault interrupt).

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.

C.2 Server RAS architecture requirements

The following must be true of all PEs and other system components that implement the Arm RAS System Architecture [9] in the base server system.

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

Within the the standard programming model:

- The size of the error counter is an IMPLEMENTATION DEFINED choice of 7 or 15 bits.
- 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. Therefore, the requirement should include guidance for making this type of choice, rather than prescribe one format.

R_{RAS_02} If the component is a memory controller or significant cache, it must implement the CFI, DUI, and UI controls described in the RAS System Architecture v1.1.

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 an critical error interrupt, the group implements a single critical error interrupt for all the records contained in the group.

R_{RAS_06} Each fault handling, error recovery, or critical error interrupt that is generated by a node that is accessible to a PE and is not implemented as a message-signaled interrupt must be connected to the base system GIC interrupt controller. Unless otherwise required to be implemented as a PPI, the interrupt is an SPI.

Note: This rule does not prevent the interrupt also being connected to, for example, a system control processor.

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 ERRADDR.AI bit 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 location is in system memory that is accessible by PEs in the base system, it is preferred if the physical address is the SPA. If the physical address captured for a location in system memory that is accessible to PEs is not the the SPA, Arm recommends that the recorded address is translated to the SPA without active firmware assistance.

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 PFA. 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 reports the status of the error record.

C.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 1.7.7.3.

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 Arm recommends 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 a software fault error, 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.

C.2.1.1 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 Arm recommends 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. Arm recommends Level 1 PE cache is classified as a large cache.

I Arm recommends that each memory controller that implements error detection, and each system cache and other large cache that can hold dirty data and implements error detection, does so to at least a SECDED or SDEC standard.

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

I Arm recommends 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.