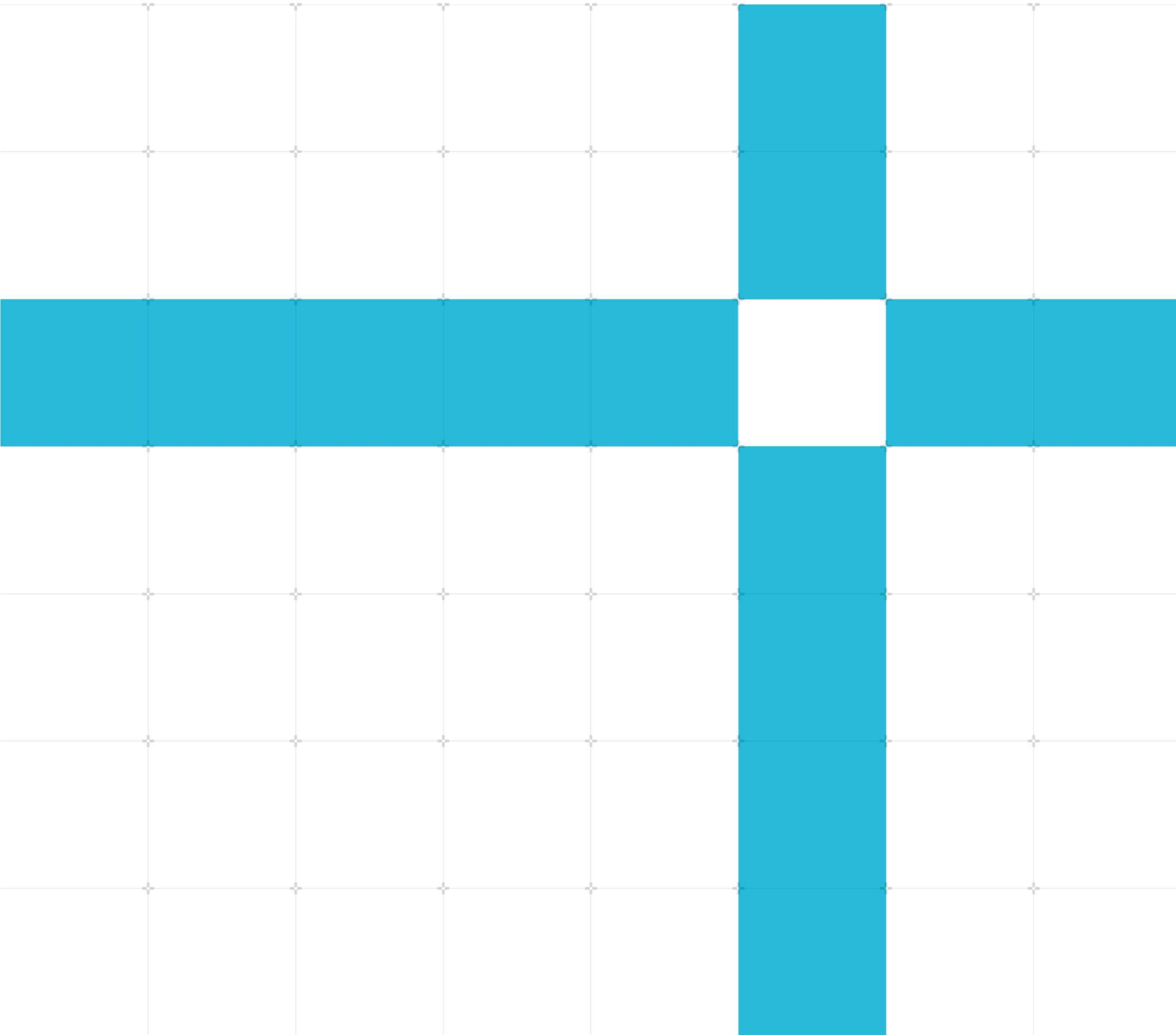
Arm® SBSA Architecture Compliance

Revision: r7p1

Test Scenario

Non-Confidential

Issue 0701-03
PJDOC-2042731200-3439



Arm SBSA Test Scenario Document

Release information

Document history

Issue	Date	Confidentiality	Change
02	05 May 2018	Non-Confidential	Changes for REL 1.0
03	20 March 2020	Non-Confidential	Changes for REL 2.3 and REL 2.4
04	30 September 2020	Non-Confidential	Changes for REL 3.0
05	27 September 2021	Non-Confidential	Changes for REL 3.1
06	20 October 2022	Non-Confidential	Changes for REL 6.1
0700-00	15 June 2022	Non-Confidential	REL 7.0 ALPHA release
0701-01	16 January 2023	Non-Confidential	REL 7.1 BETA-0 release
0701-02	28 March 2023	Non-Confidential	REL 7.1.1 BETA-1 release
0701-03	29 June 2023	Non-Confidential	REL 7.1.2 EAC release

Non-Confidential Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

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

Unrestricted Access is an Arm internal classification.

Product Status

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

Progressive terminology commitment

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

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

Web Address

www.arm.com.

Contents

1 Introduction

1.1 Product revision status

The r*mpn* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

rm Identifies the major revision of the product, for example, r*1*.

pn Identifies the minor revision or modification status of the product, for example, p*2*.

1.2 Intended audience

This document is for engineers who are verifying an implementation of Arm® Server Base System Architecture 7.1.

1.3 Conventions

The following subsections describe conventions used in Arm documents.

1.3.1 Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: **https://developer.arm.com/glossary**.

1.3.2 Typographical Conventions

Convention	Use
italic	Introduces citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace` **`bold`**	Denotes language keywords when used outside example code.
`monospace underline`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

1.4 Useful resources

This document contains information that is specific to this product. See the following resources for other relevant information.

- Arm Non-Confidential documents are available on **developer.arm.com/documentation**. Each document link in the tables below provides direct access to the online version of the document.
- Arm Confidential documents are available to licensees only through the product package.

Arm products	Document ID	Confidentiality
Arm® Server Base System Architecture 7.1	DEN0029H	Non-Confidential
Arm® Architecture Reference Manual Armv8, for Armv8-A Architecture	ARM DDI 0487H.a	Non-Confidential
Arm® System Memory Management Unit Architecture Specification	IHI0070	Non-Confidential
Arm® Architecture Reference Manual Supplement Memory System Resource Partitioning and Monitoring (MPAM), for A-profile architecture	DDI0598	Non-Confidential
Arm® CoreSight Performance Monitoring Unit Architecture	IHI0091	Non-Confidential
Arm® Reliability, Availability and Serviceability (RAS) specification, for A-profile architecture	DDI0587	Non-Confidential

Non-Arm resources	Document ID	Confidentiality
PCI Express Base Specification Revision 5.0, Version 1.0	*NA*	Non-Confidential

1.5 Feedback

Arm welcomes feedback on this product and its documentation.

1.5.1 Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

1.5.2 Feedback on content

If you have comments on content, send an email to **support-systemready-acs@arm.com** and give:

- The title Arm Base System Architecture Scenario.
- The number PJDOC-2042731200-3439
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.
- Arm also welcomes general suggestions for additions and improvements.

2 Arm Server Base System Architecture

The SBSA specifies a hardware system architecture that is based on Arm 64-bit architecture. The server system software such as operating systems, hypervisors, and firmware can rely on this architecture. It addresses PE features and key aspects of system architecture.

The primary goal is to ensure enough standard system architecture to enable a suitably built single OS image to run on all the hardware compliant with this specification. A driver-based model for advanced platform capabilities beyond basic system configuration and boot are required. However, that is outside the scope of this document. Fully discoverable and describable peripherals aid the implementation of such a driver model.

SBSA also specifies features that firmware can rely on, allowing for some commonality in firmware implementation across platforms.

2.1 SBSA ACS

The tests are divided into a hierarchy of subcategories depending on the runtime environment and the component submodules that are required for achieving the verification. The top level of the hierarchy is consistent with the target hardware subsystem which is validated by the test.

These are compliance level 3 to compliance level 7 as per SBSA specification version 7.1. A test may check for different parameters of the hardware subsystem based on the level of compliance requested.

SBSA compliance also require system to be complaint with BSA specification. Please refer https://github.com/ARM-software/bsa-acs/blob/main/docs/Arm_Base_System_Architecture_Scenario_ES.pdf for BSA tests.

The tests are classified as:

- PE
- Memory Map
- GIC
- SMMU
- Timer
- Watchdog
- PCIe
- RAS
- PMU
- MPAM

2.2 PE

Test number	Rule ID	Level	Rule	Algorithm
1	S_L3PE_01	L3	PEs must support 4KB and 64KB translation granules at stage 1 and stage 2.	For PE implementation before Armv8.5, 1. ID_AA64MMFR0_EL1 bits [43:36] should be RES0. 2. TGran4 and TGran64 fields of ID_AA64MMFR0_EL1 register should read zero. For PE implementation Armv8.5 and after, 1. ID_AA64MMFR0_EL1 bits [43:36] should not be 0. 2. In ID_AA64MMFR0_EL1 register, TGran4 field of should read 0 or 0b0001 and TGran64 should read 0. 3. In ID_AA64MMFR0_EL1 register, TGran4_2 field of should read 0b0010 or 0b0011 and TGran64_2 should read 0b0010
2	S_L3PE_02	L3	PEs must implement 16-bit ASID support	CPU system register field ID_AA64MMFR0_EL. ASIDBits must read b0010.
3	S_L3PE_03	L3	PEs must implement AArch64 at all implemented Exception levels	CPU system register ID_AA64PFR0_EL1 fields EL0, EL1, EL2 and EL3 must read non-zero value.
4	S_L3PE_04	L3	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.	Check for FEAT_LPA presence, CPU System register ID_AA64MMFR0_EL1 bits [3:0] must read b0110. If FEAT_LPA is implemented, then all peripheral addresses should be contained inside 2^48 memory map
5	S_L4PE_01	L4	All PEs must implement the RAS extension that is introduced in Armv8.2. See FEAT_RAS in [2].	CPU system register field ID_AA64PFR0_EL1.RAS must read a non-zero value.
6	S_L4PE_02	L4	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	Check whether persistent memory exists in the system from using UEFI GetMemoryMap() service. If present, CPU System register field ID_AA64ISAR1_EL1.DPB must read b0001 or b0010 indicating DC CVAP instruction support.
7	S_L4PE_03	L4	All PEs must implement FEAT_VMID16	CPU system register field ID_ AA64MMFR1_EL1.VMIDBits must read b0010.

Test number	Rule ID	Level	Rule	Algorithm
8	S_L4PE_04	L4	All PEs must implement FEAT_VHE	CPU system register field ID_ AA64MMFR1_EL1.VH must read b0001.
9	S_L5PE_01	L5	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.	CPU system register field ID_AA64MMFR2_EL1. BBM must read b0001 or b0010.
10	S_L5PE_02	L5	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].	CPU system register field ID_AA64ISAR1_EL1.APA must read a non-zero value.
	S_L5PE_03	L5	PEs that are based on Armv8.4 must implement the requirements of the CS-BSA combination C [8].	Not Implemented CS-BSA is entirely new ACS for ARM CoreSight and not included with current release.
11	S_L5PE_04	L5	All PEs must implement the Activity Monitors Extension (AMU).	CPU system register field ID_AA64PFR0_EL1.AMU must read non-zero value
12	S_L5PE_05	L5	Where export control allows, all PEs must implement cryptography support for SHA3 and SHA512. See FEAT_SHA3 and FEAT_SHA512 in [2	CPU system register ID_AA64ISAR0_EL1, SHA3 field must read b0001 and SHA2 field must read b0010 for all the PEs.
13	S_L5PE_06	L5	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].	CPU system register field ID_AA64MMFR2_EL1.FWB must read b0001.
14	S_L5PE_07	L5	All PEs must implement enhanced nested virtualization, that is provided by HCR_EL2.NV2 and the VNCR_EL2 register. See FEAT_NV2 in [2].	CPU system register field ID_AA64MMFR2_EL1. NV must read b0010.

Test number	Rule ID	Level	Rule	Algorithm
15,16	S_MPAM_PE	L5	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.	Check for FEAT_MPAM implementation, CPU System register field ID_AA64PFR0_EL1.MPAM > 0 or ID_AA64PFR1_EL1.MPAM_frac > 0 If implemented then following conditions needs to be met, 1. CPU System register MPAMIDR_EL1, field PARTID_MAX >= 16 2. CPU System register MPAMIDR_EL1, field HAS_HCR = 1 3. CPU System register MPAMIDR_EL1, field VPMR_MAX > 0 4. CPU System register MPAMIDR_EL1, field PMG_MAX >= 2 5. For Last level cache reported by PPTT ACPI Table, the memory mapped register field MPAMF_IDR.HAS_CPOR_PART of corresponding MSC node in MPAM ACPI table must read b1.
MTE Linux test	B_PE_16	L6	If FEAT_MTE (Memory Tagging Extension), is implemented, • The implementation can be full including instructions and checks, or just provide support for the instructions. • All general-purpose volatile host DRAM that can be used by an operating system for applications must support memory tagging. • Dedicated memories for accelerators, or remote memory, or non-volatile memory do not need to support it. Firmware tables will indicate the memory ranges to the OS.	Check for FEAT_MTE implementation, If implemented, test for memory tagging, 1. Enable the tagged address ABI, synchronous or asynchronous MTE tag check fault using prctl() call. 2. Allocate a page of memory with MTE enabled using mmap() and mprotect() calls. 3. Insert a randomly generated logical tag to pointer to the memory allocated above using IRG instruction. 4. Set same allocation tag to first 16 byte of memory using STG instruction. 5. Call fork () to create a child process and in child process do a memory access to any byte in next 16-byte granule using same pointer that is tagged in step 3 6. Due to mismatch in allocation and logical tag, the access made in step 5, the child process should exit with SIGSEGV exception if MTE is enabled correctly. 5. Test is PASSED if parent process sees child process exited with SIGSEGV else the test is FAILED.

Test number	Rule ID	Level	Rule	Algorithm
17	B_PE_17	L6	If PEs implement the Scalable Vector Extension (SVE) and the Statistical Profiling Extension (SPE), the PEs must implement FEAT_SPEv1p1.	If ID_AA64PFR0_EL1.SVE is implemented, then ID_AA64DFR0_EL1.PMSVer must be supported.
18	S_L6PE_02	L6	PEs must provide support for Branch Target Identification. Support is indicated by register ID_AA64PFR1_EL1.BT== b0001. See FEAT_BTI in [2].	CPU system register field ID_ AA64PFR1_EL1.BT must read b0001.
19	S_L6PE_03	L6	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	CPU system register field ID_ AA64MMFR2_EL1. E0PD must read b0001.
20	S_L6PE_04	L6	All PEs must implement FEAT_PMUv3p5 [2].	CPU system register field ID_AA64DFR0_EL1.PMUVer field >= 6 and !=0xF.
21	S_L6PE_05	L6	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].	CPU system register field ID_ AA64MMFR1_EL1.HAFDBS must read b0010.
22	S_L6PE_06	L6	PEs must provide support for enhanced virtualization traps as indicated by ID_AA64MMFR2_EL1.EVT==b0010. See FEAT_EVT in [2].	CPU system register field ID_ AA64MMFR2_EL1.EVT must read b0010.
23	B_SEC_01	L6	PEs must implement the restrictions on speculation that are introduced in the Arm v8.5 extensions to the Arm architecture and SCXTNUM_ELx registers as indicated by ID_AA64PFR0_EL1.CSV2==b0010 and ID_AA64PFR0_EL1.CSV3==b0001. See FEAT_CSV2 and FEAT_CSV3.	ID_AA64PFR0_EL1.CSV2 and ID_AA64PFR0_EL1.CSV3 must be supported.
24	B_SEC_02	L6	PEs implement the PSTATE/CPSR SSBS (Speculative Store Bypass Safe) bit and the instructions to manipulate it. See FEAT_SSBS. This is identified by ID_AA64PFR1_EL1.SSBS==b0010	ID_AA64PFR1_EL1.SSBS must be supported
25	B_SEC_03	L6	PEs implement the CSDB, SSBB, and PSSBB barriers.	ID_AA64PFR1_EL1.SSBS must be supported.
26	B_SEC_04	L6	PEs implement the FEAT_SB. This is indicated by ID_AA64ISAR1_EL1.SB== b0001.	ID_AA64ISAR1_EL1.SB must be supported.
27	B_SEC_05	L6	PEs implement the CFP RCTX, DVP RCTX, CPP RCTX instructions to restrict use of information gathered through control flow, data value prediction, or cache prefetch prediction, from affecting speculative execution. See the FEAT_SPECRES. Support for the instructions is indicated by ID_AA64ISAR1_EL1.SPECRES==b0 001.	ID_AA64ISAR1_EL1.SPECRES must be supported.

Test number	Rule ID	Level	Rule	Algorithm
28	S_L7PE_01	L7	PEs must implement fine grained trap support as indicated by ID_AA64MMFR0_EL1.FGT== b0001. See FEAT_FGT.	CPU System register field ID_AA64MMFR0_EL1.FGT must read b0001.
29	S_L7PE_02	L7	PEs must implement the enhanced counter virtualization functionality as indicated by ID_AA64MMFR0_EL1.ECV == b0010. See FEAT_ECV.	CPU System register field ID_AA64MMFR0_EL1.ECV must read b0010.
30	S_L7PE_03	L7	PEs must implement the enhancements to the Activity Monitor Unit (AMU) as indicated by ID_AA64PFR0_EL1.AMU == b0010. See FEAT_AMUv1p1.	CPU System register field ID_AA64PFR0_EL1.AMU must read b0010.
31	S_L7PE_04	L7	PEs must implement the Advanced SIMD Int8 matrix multiply extension as indicated by: ID_AA64ISAR1_EL1.I8MM == b0001. See FEAT_I8MM.	CPU System register field ID_AA64ISAR1_EL1.I8MM must read b0001.
32	S_L7PE_05	L7	PEs must implement the BFLOAT16 extension. See FEAT_BF16.	CPU System register field ID_AA64ISAR1_EL1.BF16 must read b0001.
33	S_L7PE_06	L7	PEs must implement the EnhancedPAC2 and FPAC extensions as indicated by ID_AA64ISAR1_EL1.APA == b0101. See FEAT_PAuth2.	CPU System register field ID_AA64ISAR1_EL1.APA must read b0101.
34	S_L7PE_07	L7	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.	Check for SVE implementation, CPU system register field ID_AA64PFR0_EL1.SVE must read b0001. If SVE implemented, CPU system register ID_AA64ZFR0_EL1.I8MM must read b0001.
35	S_L7PE_08	L7	Arm recommends the implementation of the data gathering hint feature as indicated by ID_AA64ISAR1_EL1.DGH == b0001. See FEAT_DGH.	CPU System register field ID_AA64ISAR1_EL1.DGH must read b0001.
36	S_L7PE_09	L7	Arm recommends the implementation of the WFE fine tuning delay feature as indicated by ID_AA64MMFR0_EL1.TWED == b0001. See FEAT_TWED.	CPU System register field ID_AA64MMFR0_EL1.TWED must read b0001.
37	S_L7PE_10	L7	Arm strongly recommends that enhanced PAN feature is implemented, which is introduced in Arm v8.7-A but can be implemented from v8.1-A. Support for enhanced PAN is indicated by ID_AA64MMFR1_EL1.PAN == b0011.	CPU System register field ID_AA64MMFR1_EL1.PAN must read b0011.
	S_L7TME_1	L7	All PEs must have the same value of ID_AA64ISAR0_EL1.TME.	Not Implemented TME rules implementation for this release is waived off due to lack of hardware supporting the feature.

Test number	Rule ID	Level	Rule	Algorithm
	S_L7TME_2	L7	The Latency of starting and committing a transaction must not be higher than the latency of the code sequence that Arm recommends for acquiring and releasing a spinlock	Not Implemented TME rules implementation for this release is waived off due to lack of hardware supporting the feature.
	S_L7TME_3	L7	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	Not Implemented TME rules implementation for this release is waived off due to lack of hardware supporting the feature.
	S_L7TME_4	L7	Arm recommends using the hardware cache coherency facilities of the processor to detect transactional conflicts. This is also known as eager conflict detection	Not Implemented TME rules implementation for this release is waived off due to lack of hardware supporting the feature.
	S_L7TME_5	L7	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	Not Implemented TME rules implementation for this release is waived off due to lack of hardware supporting the feature.
1301	S_L7ENT_1	L7	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.	SBSA ACS utilize NIST STS package for randomness testing. Please refer Arm_SBSA_NIST_User_Guide to run NIST STS tests.

2.3 Memory Map

Test number	Rule ID	Level	Rule	Algorithm
101	S_L3MM_01	L3	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.	Check whether all peripheral base addresses are 64KB apart from each other.
101	S_L3MM_02	L3	Peripherals that will be assigned to different virtual machines will be situated within different 64KB regions of memory	Covered by test < S_L3MM_01>

2.4 GIC

Test number	Rule ID	Level	Rule	Algorithm
201	S_L3GI_01	L3	A base server system must implement an interrupt controller that is compliant with the GICv3 or higher architecture	Check GIC version is 3. or greater than 3
202	S_L5PP_01	L5	In addition to the PPI assignment specified in Arm BSA [5], the following PPIs are reserved by the SBSA specification	Check none of reserved interrupt is mentioned in ACPI tables
809	S_L3GI_02	L3	All MSI and MSI-X targeting hypervisor and operating system software must be mapped to LPI.	Pull each discovered PCI device and its list of MSI(X) vectors and Check whether every vector IRQ number is an LPI or not.
	S_L5GI_01	L5	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	Not Implemented No generic way to find all the interrupt ctrl defined in system if not included in ACPI table and whether an interrupt controller requires a platform specific kernel driver.

2.5 SMMU

Test number	Rule ID	Level	Rule	Algorithm
-	S_L3SM_01	L3	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].	Not Implemented Which requester are OS visible and have TLB in a system are IMPDEF and no generic way to obtain that information
301	S_L4SM_01	L4	Stage 1 System MMU functionality must be provided by a System MMU that is compliant with the Arm SMMUv3, or higher, architecture revision.	The SMMU memory mapped register field SMMU_AIDR.ArchMajorRev ≥ 3 indicating SMMU is compliant with SMMUv3 or higher. And memory mapped register field SMMU_IDR0.S1P must read b1 indicating Stage 1 support.
301	S_L4SM_02	L4	Stage 2 System MMU functionality must be provided by a System MMU that is compliant with the Arm SMMUv3, or higher, architecture revision.	The SMMU memory mapped register field SMMU_AIDR.ArchMajorRev ≥ 3 indicating SMMU is compliant with SMMUv3 or higher. And memory mapped register field SMMU_IDR0.S2P must read b1 indicating Stage 2 support.
	S_L4SM_03	L4	The integration of the System MMUs is compliant with the rules in SMMUv3 integration appendix in Arm BSA [5].	SMMU_01: The SMMU memory mapped register field SMMU_IDR0.COHACC must read b1. Covered by BSA test 354 SMMU_02: Not implemented for current release.
302	S_L5SM_01	L5	SMMU implementations must be compliant with the Arm SMMUv3.2 architecture revision or higher.	The SMMU memory mapped registers fields must satisfy SMMU_AIDR.ArchMajorRev ≥ 3 and SMMU_AIDR.ArchMinorRev ≥ 2.
302	S_L5SM_02	L5	SMMU implementations must provide level 1 or level 2 support for translation table resizing.	The SMMU memory mapped registers fields must satisfy SMMU_AIDR.ArchMajorRev ≥ 3 and SMMU_AIDR.ArchMinorRev ≥ 2.

Test number	Rule ID	Level	Rule	Algorithm
305	S_L5SM_03	L5	An SMMUv3.2 implementation must support the MPAM extension if the requests it serves access MPAM controlled resources.	1. Check if there are any MPAM-controlled memory resources (Read out: ID_AA64PFR0_EL1. MPAM). 2. If the architecture revision of the SMMU is at least v3.2, check for MPAM support on the SMMU side (SMMU_IDR3.MPAM == '1' and SMMU_MPAMIDR.PARTID_MAX !=
313	S_L6SM_01	L6	The SMMU must implement coherent access to in memory structures, queues, and page tables as indicated by SMMU_IDR0.COHACC = 0b1.	The SMMU memory mapped register field SMMU_IDR0.COHACC must read b1.
306	S_L6SM_02	L6	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.	The SMMU memory mapped register field SMMU_IDR0.HTTU must read b10.
307	S_L6SM_03	L6	The SMMU must support Message Signaled Interrupts as indicated by SMMU_IDR0.MSI = 0b1	The SMMU memory mapped register field SMMU_IDR0. MSI must read b1.
315	S_L7SM_01	L7	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.	All PCIe root ports and devices reported as named components in IORT ACPI table should be behind an SMMU, if there are DMA capable. 1. DMA capability of PCIe root ports and named components can be determined by reading CCA attribute field of node in IORT ACPI table. 2. Device is said to be behind an SMMU if output reference in IORT node's ID mapping points to an SMMU.

Test number	Rule ID	Level	Rule	Algorithm
316	S_L7SM_02	L7	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	1. Search for ETR implementation in system's ACPI namespace using its unique HID "ARMHC501". 2. If ETR(s) is/are present, check named component entries in IORT ACPI table and find nodes whose "Device object name" field matches with full path to ETR node in namespace. 3. Check if an ETR is behind an SMMU using output reference field in corresponding named component node in IORT ACPI table. 4. If ETR is not behind an SMMU, then check for CATU in the path of ETR.
314	S_L7SM_03	L7	SMMU must implement the SMMUv3 Performance Monitors Extension	1. Parse IORT ACPI table for PMCG nodes and use PMCG node address field to check whether it maps to a SMMU. 2. All SMMUs reported by IORT ACPI table must have at least one PMCG associated with it.
314	S_L7SM_04	L7	All SMMU Performance Monitor Counter Groups must implement at least four counters	For each PMCG associated with an SMMU, SMMU PMCG memory mapped register field SMMU_PMCG_CFGR.NCTR must read value greater than 4.

2.6 Timer

Test number	Rule ID	Level	Rule	Algorithm
	S_L5TI_01	L5	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.	Not implemented. RW access to CNTControlBase is allowed only through EL3. The SBSA ACS is run on Non-secure mode. Cannot read the register value from the tests.

2.7 Peripheral

Test number	Rule ID	Level	Rule	Algorithm
601	S_L4PCI_2	L4	There must be no OS observable use of PCIe Enhanced Allocation	Pass if No EA Capability is present or check if "Enable" bit in Entry Type register is 0

2.8 Watchdog

Test number	Rule ID	Level	Rule	Algorithm
701	S_L6WD_01	L6	The generic watchdog revision must be v1, W_IIDR==0001b	Check watchdog revision for all non-secure watchdogs is 1

2.9 PCIe

Test number	Rule ID	Level	Rule	Algorithm
	S_L4PCI_1	L4	All peripherals that are intended for assignment to a virtual machine or a user space device driver must be based on PCI Express	Manual Verification All peripherals identified in ACPI SPCR table must have bus number non-zero
805	PCI_MM_01	L6	All systems must support mapping PCI Express memory space as device memory	Read BAR 0 details for the device and * Map mmio space to ARM device memory in MMU page tables. Write predefined data to BAR space and read it back
-	PCI_MM_02	L6	All systems must support mapping PCI Express memory space as non-cacheable memory	Covered as part of 805
-	PCI_MM_03	L6	When PCI Express memory space is mapped as normal memory, the system must support unaligned accesses to that region	Covered as part of 805
858	RE_BAR_1	L6	All BAR registers in an RCiEP must be writeable and readable as per the PCIe specification	Read and write to all the BAR registers
-	RE_BAR_2	L6	Dynamic re-programming of these BAR registers must be allowed	Covered as part of 805

Test number	Rule ID	Level	Rule	Algorithm
834	RE_BAR_3	L6	A RCiEP must not support I/O space claimed through BARs	Check if BAR registers support MMIO
-	IE_BAR_1	L6	All BAR registers in an i-EP endpoint and i-EP Root Port must be writeable and readable as per the PCIe specification.	Covered as part of 858
-	IE_BAR_2	L6	Dynamic re-programming of these BAR registers must be allowed	Covered as part of 805
-	IE_BAR_3	L6	A i-EP Endpoint, and its i-EP Root Port, must not support IO space claimed through BARs.	Covered as part of 834
835	RE_RST_1	L6	RCiEP must have Function Level Reset (FLR) support	Check for FLR capability, if yes, Initiate FLR by setting the FLR bit. The device must be accessible after max FLR period
835	IE_RST_1	L6	i-EP Endpoint must have Function Level Reset (FLR) support	Check for FLR capability, if yes, Initiate FLR by setting the FLR bit. The device must be accessible after max FLR period
857	IE_ACS_1	L6	ACS capability must be present in the i-EP endpoint functions if the i-EP Endpoint is a multi-function device and supports peer to peer traffic between its functions. It must comply with the PCIe specification on specific ACS access controls that must be supported. If the i-EP Endpoint has ACS capability, then it must have AER capability for reporting ACS violation errors	Check for ACS capability registers
856	IE_ACS_2	L6	The i-EP Root Port must have ACS capability if the i-EP Endpoint can send transactions to a peer endpoint. It must comply with the PCIe specification on specific ACS access controls that must be supported. If the i-EP Root Port has ACS capability, then it must have AER capability for reporting ACS violation errors	Check for ACS capability registers
-	RE_ACS_1	L6	ACS capability must be present in the RCiEP if the RCiEP is a multi-function device and supports peer to peer traffic between its functions. It must comply with the PCIe specification on specific ACS access controls that must be supported.	Covered as part of 857
-	RE_ACS_2	L6	If the RCiEP has ACS capability, then it must have AER capability for reporting ACS violation errors.	Covered as part of 857
860	RE_PCI_1	L6	A RCiEP must obey all the rules that are specified in Section 1.3.2.3 of the PCIe 5.0 specification [1].	Check for capability registers

Test number	Rule ID	Level	Rule	Algorithm
859	RE_PCI_2	L6	The RCEC must obey all the rules that are specified in Section 1.3.4 of the PCIe 5.0 specification [1].	Check for capability registers
852, 902, 903	RE_SMU_2		PCIe ATS capability must be supported if the RCiEP has a software visible cache for address translations.	ATS capability should be supported. Initialize DMA master and memory descriptors and enable SMMU Send an ATS Translation Request for the VA, Get ATS Translation Response, Compare Translated Addr with Physical Address from the Mappings, Configure Exerciser to issue subsequent DMA transactions, Trigger DMA from input buffer to exerciser memory and Trigger DMA from exerciser memory to output buffer
	RE_SMU_4	L6	If the RCiEP supports PASIDs, the PASID is used as SubStreamID as specified in the SMMU architecture specification	For each exerciser behind an SMMU, * Create a mapping of 1 IOVA region to 2 PA regions, via SMMU * Each of the 2 mappings is identified by a distinct PASID * Confiugure exerciser DMA engine to access IOVA region base. * Configure exerciser to execute DMA with TEST_PASID1 in transactions, Accesses must PA region 1. * Configure exerciser to execute DMA with TEST_PASID2 in transactions, Accesses must PA region 2. Covered as part of BSA
820, 821, 830, 831, 832	IE_REG_1	L6	All type 0 header registers must be implemented for i-EP Endpoint. The registers must be implemented as described in PCIe specification.	Check for capability registers
824, 825, 826, 827, 833, 837, 839	IE_REG_2	L6	The registers mandated in Section 7.5.3 of the PCIe specification [1] for an endpoint that is not a RCiEP must be implemented for i-EP Endpoint	Check for capability registers
820, 822, 830, 831, 832	IE_REG_3	L6	All type 1 header registers must be implemented for i-EP Root Port. The registers must be implemented as described in the PCIe specification.	Check for capability registers
823,824, 825,826, 827,833, 838	IE_REG_4	L6	The registers mandated in Section 7.5.3 of the PCIe specification [1] for Root Ports must be implemented for Root Port	Check for capability registers
820, 821, 830, 831, 832	RE_REG_1	L6	All type 0 header registers must be implemented for RCiEP and RCEC.	Check for capability registers

Test number	Rule ID	Level	Rule	Algorithm
828, 829	RE_REG_2	L6	All registers of the PCI power management capability must be implemented for RCiEP and RCEC	Check for capability registers
824, 825, 826, 827	RE_REG_3	L6	The registers that are specified in Section 7.5.3 of the PCIe specification [1] for all devices must be implemented for RCiEP	Check for capability registers
841	RE_INT_1	L6	If the RCiEP supports interrupt generation, the RCiEP must support MSI or MSI-X interrupt generation	Check for capability register
841	IE_INT_1	L6	i-EP Endpoint must generate only MSI or MSI-X interrupts	Check for capability register
842	RE_PWR_1	L6	RCiEP must have D state support and must have PCI Power management capability as specified in the PCIe specification	Check for capability register
904	RE_ORD_1	L6	The RCiEP must obey PCIe ordering rules for the configuration and BAR mapped memory spaces when accessed in the inbound direction, towards the RCiEP.	Map mmio and config space to ARM device(nGnRnE and nGnRE) memory in MMU page tables and Perform Transactions on incremental aligned address and on same address
-	RE_ORD_2	L6	PCIe ordering rules must be obeyed while sending out completions for configuration space and BAR mapped memory space accesses	Covered as part of 904
-	IE_ORD_1	L6	The i-EP must obey PCIe ordering rules for the configuration and BAR mapped memory spaces when accessed in the inbound direction, towards the i-EP.	Covered as part of 904
-	IE_ORD_2	L6	PCIe ordering rules must be obeyed while sending out completions for configuration space and BAR mapped memory space accesses.	Covered as part of 904
	S_PCIe_01	L7	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	Covered as part of PCIe test suite
861	S_PCIe_02	L7	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.	Do 1B and 2B aligned/unaligned read/write on Pre/Non-Pre address space for all functions

Test number	Rule ID	Level	Rule	Algorithm
905	S_PCIe_03	L7	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.	Use to monitor if the 2/4/6 B writes are done as 2/4/8 B writes
908	S_PCIe_04	L7	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.	We can check if 2-byte DMA write from exerciser modifies only 2B in memory. But we cannot monitor the transaction. But with p2p, we can enable this.
	S_PCIe_05	L7	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	Not Implemented unfeasible to determine if the PHY LTSSM is disabled.

Test number	Rule ID	Level	Rule	Algorithm
906	PCI_ER_01	L7	Root Port must support the Advanced Error Reporting feature as described in section 6.2 and 7.8.4 of [1].	1. Enable errors in Status register for all RPs 2. Enable Error reporting in the device control register of PCI-ECS for all RPs and Eps 3. For each EPs, toggle the root error command bits of the AER cap register and check if error is reported and not reported in root port 4. For each EPs, set toggle the error mask enable/disable and toggle the error severity fatal/non-fatal for uncorrectable error and check if it is received by the RP 5. Check in RP's AER cap - root error status register if the error bit are set/unset when generated based on the config 6. Check in RPs device status register of PCIE-ECS if fatal/non-fatal/UR bits are set/unset when errors are generated based on config 7. Check if the requester ID in the error source identification register of the AER CAP has the correct ID 8. Check if RP implements MSI and enable/disable error reporting in root error command register of AER cap and check if interrupts are being generated/not-generated based on config
-	PCI_ER_02	L7	The Root Port must implement the MSI capability.	Covered by test 906
-	PCI_ER_03	L7	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].	Covered by test 906
-	PCI_ER_04	L7	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].	Covered by test 906

Test number	Rule ID	Level	Rule	Algorithm
907	PCI_ER_05	L7	The Root Port must implement Downstream Port Containment feature. See section 6.2.10 in Reference [1].	1. DPC is disabled by default and cannot be triggered unless enabled by software using the DPC Trigger Enable field. 2. When the DPC Trigger Enable field is set to 01b, DPC is enabled and is triggered when the Downstream Port detects an unmasked uncorrectable error or when the Downstream Port receives an ERR_FATAL Message 3. When the DPC Trigger Enable field is set to 10b, DPC is enabled and is triggered when the Downstream Port detects an unmasked uncorrectable error or when the Downstream Port receives an ERR_NONFATAL or ERR_FATAL Message 4. When DPC is triggered due to receipt of an uncorrectable error Message, the Requester ID from the Message is recorded in the DPC Error Source ID Register, and that Message is discarded and not forwarded Upstream 5. When DPC is triggered, the Downstream Port immediately Sets the DPC Trigger Status bit and DPC Trigger Reason field to indicate the triggering condition and disables its Link by directing the LTSSM to the Disabled state 6. Once the LTSSM reaches the Disabled state, it remains in that state until the DPC Trigger Status bit is Cleared. To ensure that the LTSSM has time to reach the Disabled state or at least to bring the Link down under a variety of error conditions, software must leave the downstream Port in DPC until the Data Link Layer Link Active bit in the Link Status Register reads 0b 7. When the DPC Trigger, in RP the Status bit is Set and the DPC RP Busy bit is Set, software must leave the Root Port in DPC until the DPC RP Busy bit reads 0b

Test number	Rule ID	Level	Rule	Algorithm
-	PCI_ER_06	L7	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].	Covered by test 907
823, 824, 825, 826, 827, 833, 838, 863	IE_REG_4	L6	The registers mandated in Section 7.5.3 of the PCIe specification [1] for Root Ports must be implemented for Root Port. The registers must be implemented as described in Table 34. Slot capabilities register - All bits in these registers must be set to 0. Slot Control register - For the Root Port, Data Link Layer State Changed Enable bit must be implemented as per PCIe specification. All other bits in this register must be set to 0. Slot Status Register - For i-EP Root Port, Data link layer state changed, and Presence detect state register fields implement as described in PCIe specification. All other bits in this register must be set to 0.	Register checks. Check the value and access attribute for each of the Slot Capabilities, Slot Control and Slot Status register fields.

2.9 RAS

Test number	Rule ID	Level	Rule	Algorithm
	S_RAS_01	L6	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.	Not Implemented Will require reading the generic counter timer, which needs secure access.

Test number	Rule ID	Level	Rule	Algorithm
	S_RAS_02	L6	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.	Not Implemented. This is a recommendation rule.
1209	S_L7RAS_1	L7	For containable errors, error exceptions on loads from Normal memory must be taken as synchronous Data Abort exceptions.	1. Check for All Memory Controller RAS Nodes. 2. Allocate an address accessible to PE. 3. Setup/Inject containable error in an implementation defined way. 4. Check for External Abort.
	S_L7RAS_2	L7	Errors that are signaled to a PE on speculative accesses must not generate Abort exceptions.	Not Implemented The rule covers the errors signaled to PE on speculative access, its unfeasible from software to guarantee that access will be speculative.
1201	RAS_01	L7	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.	1. Check for All Memory Controller or cache RAS Nodes. 2. Read the FR register of the first error record. ERR<n>FR, Error Record Feature Register, Read ERR<n>FR.CEC
1202	RAS_02	L7	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.	1. Check for All Memory Controller RAS Nodes. 2. Read the CTLR register of the first error record. ERR<n>CTLR, Error Record Control Register ERR<n>CTLR.CFI, ERR<n>CTLR.DUI, ERR<n>CTLR.UI

Test number	Rule ID	Level	Rule	Algorithm
1203	RAS_03	L7	Each error record group implements a single fault handling interrupt for all the records that are contained in the group.	1. Check for All RAS Nodes who has same error record base address. 2. Check if the node implements FHI (From AEST Table) 3. for all nodes Check for the corresponding interrupt ID from AEST table node Interrupt array for FHI. 4. Value should be same within a group.
1204	RAS_04	L7	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.	1. Check for All RAS Nodes who has same error record base address. 2. Check if the node implements ERI (From AEST Table) 3. for all nodes Check for the corresponding interrupt ID from AEST table node Interrupt array for ERI. 4. Value should be same within a group.
	RAS_05	L7	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.	Not Implemented AEST table only exposes EHI and FHI information and not CEI information.
1205	RAS_06	L7	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.	1. Check for All RAS Nodes accessible to PE. 2. Get the node FHI/ERI information from AEST Table. 3. Check if the Interrupt ID is SPI/PPI or not. 4. Fail the test if ID is not SPI/PPI. 5. Setup/Inject error for FHI/ERI. 6. Check interrupt should be raised.

Test number	Rule ID	Level	Rule	Algorithm
1206	RAS_07	L7	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.	1. Iterate through memory controller RAS nodes reported in AEST ACPI table. 2. Fetch base address for each proximity domain reported by AEST table. 3. Check if the fetched address is accessible to PE by allocating the memory. Skip the RAS memory controller (MC) node if memory not accessible, else continue with below steps. 4. Program the system to setup and inject an error at fetched address in platform defined way. 5. Iterate through implemented error records for the current RAS MC node and check if the node records a valid address syndrome as indicated by ERRSTATUS.AV bit. Skip the current RAS MC node if no error record captured the valid address syndrome. 6. For the error record which captures valid address syndrome, fetch the addressing mode indicated by AEST ACPI table. 7. The fetched addressing mode value and ERRADDR.AI bit value for the error record should match for compliance with the rule
1207	RAS_08	L7	If an error record of the component is accessible through an error record group, then the ERRGSR reports the status of the error record.	1. Check for All RAS Nodes. 2. Check the node interface array (From AEST Table). 3. Check for the status reporting field from AEST table node Interface array.
	RAS_10	L7	Where the PCIe standard, Arm architecture, or other standard defines a rule or sets a convention for a software fault at a device, that rule or convention must be followed.	This rule will be covered by PCIe error related tests.

Test number	Rule ID	Level	Rule	Algorithm
1208	RAS_11	L7	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.	1. Access empty space in between UART memory layout reserved space 2. Check write are ignored and read does not results into exception. 3. Fail the test if Exception is raised.
-	RAS_12	L7	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.	Covered by test 1208

Test number	Rule ID	Level	Rule	Algorithm
1210	SYS_RAS_1	L7	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.	Check for NUMA node memory information in RAS2 ACPI table. For each memory proximity domain mentioned in RAS2 table should support patrol scrubbing feature.
1211, 1212	SYS_RAS_2	L7	The system must support the storage and forwarding of poisoned values.	Check if system supports poison generation through pal API. 1. Get Error Record Index for Memory controller RAS Node. 2. Get Interrupt information from AEST table and install the handlers. 3. Program system to Setup/Inject error in platform defined way. 4. Check Error Record Status Register is updated for MC/PE RAS Nodes. 5. Check for Poison field information in ERR_STATUS if Poison Generation is supported. 6. If poison generation & forwarding is not supported then check for External abort.

Test number	Rule ID	Level	Rule	Algorithm
1211	SYS_RAS_3	L7	When an uncorrectable error is detected, a component within the system may not be able to generate poison, propagate 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.	Check if system supports poison generation through pal API. 1. Get Error Record Index for Memory controller RAS Node. 2. Get Interrupt information from AEST table and install the handlers. 3. Program system to Setup/Inject error in platform defined way. 4. Check Error Record Status Register is updated for MC/PE RAS Nodes. 5. Check for Poison field information in ERR_STATUS if Poison Generation is supported. 6. If poison generation & forwarding is not supported then check for External abort.

2.10 PMU

- Please refer to User Guide on mapping of PMU PE events rule in ACS code.

Test number	Rule ID	Level	Rule	Algorithm
	PMU_PE_01	L7	The PE implements the Performance Monitors Extension	Read register ID_AA64DFR0_EL1.PMUVer, bits [11:8] and it should not be Zero Covered by BSA ACS test 9
1101	PMU_PE_02	L7	The PMU overflow signal from each PE must be wired to a unique PPI interrupt with no intervening logic.	1. Get PMU overflow PPI interrupt number from MADT table 2. Register interrupt handler with PMU PPI interrupt 3. Generate PMU overflow interrupt by i. Reset PMINTENCLR_EL1 & PMOVSCLR_EL0 to 0xFFFFFFFF ii. Set PMCR_EL0.bit0 to 1 iii. Set PMINTENSET_EL1.bit0 to 1 iv. Set PMOVSSET_EL0.bit0 to 1 4. Check interrupt is generated or not

Test number	Rule ID	Level	Rule	Algorithm
1102	PMU_PE_03	L7	Each PE must implement a minimum of six PMU event counters and the PMU cycle counter.	Read PMCR_EL0 Bits 15:11 for Number of counters and should be greater than 5.
1106	PMU_SPE	L7	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 documentation of the mappings of the Data Source values to data sources. • If the SPE implementation includes IMPLEMENTATION DEFINED packets or IMPLEMENTATION DEFINED events in the events packet, the implementation provides public documentation of these packets and events.	1. Check SPE is supported, ID_AA64DFR0_EL1.PMSVer, bits [35:32] > 0 2. If SPE is supported, PEs must support Hardware management of the Access Flag and dirty state as indicated by PMBIDR_EL1.F == 01b.
PMU Linux test	PMU_EV_01	L7	PEs in the base server system must implement the PMU events that are listed below to measure IPC. 0x0008 INST_RETIRED Instruction architecturally executed 0x0011 CPU_CYCLES Cycles	For INST_RETIRED 1. To check support INST_RETIRED event, PMCEID0_EL0<bit 0x8> should be 1. 2. Enable PMU user access by programming PMUSERENR_EL0.EN to 1. 3. Program PMSELR_EL0 with counter number. 4. Program PMXEVTYPER_EL0 with event id(0x08). 5. Set the bitmask in PMCNTENSET to enable cycle counter and all the implemented event counters. 6. Start counting by setting PMCR.E after clearing all the counters by setting PMCR.P and PMCR.C 7. Call stimulus to generate event. 8. In stimulus, run a for loop with add instruction (e.g.: a+b) . 9. Stop counting by resetting PMCR.E to disable all the implemented event counters. 10. Read PMXEVCNTR_EL0 by selecting counters in PMSELR_EL0 and check whether the counter is moved or not. 11. Repeat steps 7 to 10 with different scaling factors (SMALL, MEDIUM,

Test number	Rule ID	Level	Rule	Algorithm
				LARGE) and make sure the counts moved accordingly. For CPU_CYCLES 1. To check support CPU_CYCLES event, PMCEID0_EL0<bit 0x11> should be 1 2. Program PMU counter to monitor CPU_CYCLES event. 3. Use same algorithm for INST_RETIRED (steps 7 -11) to generate the event and monitor counter.
PMU Linux test	PMU_EV_02	L7	PEs in the base server system must implement the PMU events listed below, for each PE local cache and the last level cache, to measure cache effectiveness 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 0x0037 LL_CACHE_MISS_RD Last Level cache memory read miss 0x0039 L1D_CACHE_LMISS_RD Level 1 data cache long-latency read miss	For INST_RETIRED Refer to PMU_EV_01, For MEM_ACCESS 1. Read PMCEID0_EL0 <bit 0x13> to check support of MEM_ACCESS event and should be 1. 2. Program PMU counter to monitor MEM_ACCESS event. 3. Call stimulus to generate event 4. In stimulus, allocate 2 memory regions and do memcpy. 5. Read PMU counter 6. Repeat steps 3 to 5 with different scaling factors (SMALL, MEDIUM, LARGE) and make sure the counters moved accordingly. For MEM_ACCESS_RD 1. Check the MEM_ACCESS_RD event support & get event number from PAL API should be filled by the vendor 2. Program PMU counter to monitor MEM_ACCESS_RD event. 3. Use same algo for MEM_ACCESS to generate and monitor this event. For Lx Data Cache related events 4. Check the required data cache event support & get event number from PAL API should be filled by the vendor 5. Program PMU counter to monitor required data cache event. 6. Use same algo for MEM_ACCESS to generate and monitor this event. For Lx Instruction Cache related events

Test number	Rule ID	Level	Rule	Algorithm
			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	7. Check the required instruction cache event support & get event number from PAL API should be filled by the vendor 8. Program PMU counter to monitor required instruction cache event. 9. Generate instruction workload
PMU Linux test	PMU_EV_03	L7	The events that are listed below measure TLB effectiveness, for each applicable level of PE TLB: 0x0013 MEM_ACCESS Data memory access 0x0025 L1D_TLB Level 1 data TLB access 0x0026 L1I_TLB Level 1 instruction TLB access 0x0034 DTLB_WALK Data TLB access with at least one translation table walk 0x0035 ITLB_WALK Instruction TLB access with at least one translation table walk	For MEM_ACCESS refer to PMU_EV_02 For TLB events refer to PMU_EV_02 cache workload
	PMU_EV_04	L7	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.	Not Implemented No Demand access event defined in current SBSA 7.0 Specification.
PMU Linux test	PMU_EV_05	L7	The events listed in the following table must be implemented for cycle accounting: 0x0011 CPU_CYCLES Cycle	For CPU_CYCLES Refer PMU_EV_01.

Test number	Rule ID	Level	Rule	Algorithm
			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	
PMU Linux test	PMU_EV_06	L7	The events in the following table must be implemented for Section B.3.3: 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	For CPU_CYCLES Refer to PMU_EV_0 For OP_ RETIRED 1. Read PMCEID1_EL0 <bit 0x1A> to check support of OP_ RETIRED event and should be 1 2. Program PMU counter to monitor OP_ RETIRED event. 3. Refer algorithm of INST_RETIRED to generate & monitor event For OP_ SPEC 1. Read PMCEID1_EL0 <bit 0x1B> to check support of OP_SPEC event and should be 1 2. Program PMU counter to monitor OP_ SPEC event. 3. Refer algorithm of OP_RETIRED to generate & monitor event
PMU Linux test	PMU_EV_07	L7	The events listed below must be implemented for measuring workload: 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: 0x80C0 FP_SCALE_OPS_SPEC Scalable floating-point element operations speculatively executed	For FP_FIXED_OPS_SPEC 1. Check the FP_FIXED_OPS_SPEC event support & get event number from PAL API should be filled by the vendor 2. Program PMU counter to monitor FP_FIXED_OPS_SPEC event. 3. Call stimulus to generate the event 4. In stimulus, define 2 variables with data type of float32x2_t 5. Add these in a loop using neon instruction float32x2_t vadd_f32 (float32x2_t, float32x2_t), and loop count will vary as per scaling factor 6. Read PMU counter. 7. Repeat steps 3 to 6 with different scaling factors (SMALL, MEDIUM,

Test number	Rule ID	Level	Rule	Algorithm
			0x80C8 INT_SCALE_OPS_SPEC Scalable integer element operations speculatively executed	LARGE) and make sure the counters moved accordingly. For INT_FIXED_OPS_SPEC 1. Check the INT_FIXED_OPS_SPEC event support & get event number from PAL API should be filled by the vendor 2. Program PMU counter to monitor INT_FIXED_OPS_SPEC event. 3. Call stimulus to generate the event 4. In stimulus, define 2 variables with data type of uint32x2_t 5. Add these in a loop using neon instruction uint32x2_t vadd_u32 (uint32x2_t, uint32x2_t), and loop count will vary as per scaling factor 6. Read PMU counter. 7. Repeat steps 3 to 6 with different scaling factors (SMALL, MEDIUM, LARGE) and make sure the counters moved accordingly. For SVE related instructions: Not Implemented. No server platform to test.
PMU Linux test	PMU_EV_08	L7	The events listed below are used to measure Section B.3.5 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; mis predicted 0x8110 BR_IMMED_PRED_RETIRED Branch instruction architecturally executed, predicted immediate	For PC_WRITE_RETIRED 1. Read PMCEID0_EL0 <bit 0xC> to check support of PC_WRITE_RETIRED event and should be 1 2. Program PMU counter to monitor PC_WRITE_RETIRED event. 3. Call stimulus to generate event 4. In stimulus, execute any branch statement (if-else) multiple number of times based on scaling factor to change the flow of execution and change PC. 5. Read PMU counter. 6. Repeat steps 3 to 5 with different scaling factors (SMALL, MEDIUM, LARGE) and make sure the counters moved accordingly. • For BR_IMMED_RETIRED 1. Read PMCEID0_EL0 <bit 0xD> to check support of BR_IMMED_RETIRED and should be 1. 2. Program PMU counter to monitor BR_IMMED_RETIRED. 3. Call stimulus to generate event.

Test number	Rule ID	Level	Rule	Algorithm
			0x8111 BR_IMMED_MIS_PRED_RETIRED Branch instruction architecturally executed; mis predicted immediate 0x8112 BR_IND_PRED_RETIRED Branch instruction architecturally executed, predicted indirect 0x8113 BR_IND_MIS_PRED_RETIRED Branch instruction architecturally executed; mis predicted indirect 0x8114 BR_RETURN_PRED_RETIRED Branch instruction architecturally executed, predicted procedure return 0x8115 BR_RETURN_MIS_PRED_RETIRED Branch instruction architecturally executed; mis predicted 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; mis predicted 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, mis predicted 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; mis predicted branch, not taken 0x811C BR_PRED_RETIRED Branch instruction architecturally executed, predicted branch 0x811D BR_IND_RETIRED Branch instruction architecturally executed, indirect branch	4. In stimulus, call a function (BL instruction) multiple number of times based on scaling factor, 5. Read PMU counter. 6. Repeat steps 3 to 5 with different scaling factors (SMALL, MEDIUM, LARGE) and make sure the counters moved accordingly. • For BR_RETURN_RETIRED 1. Read PMCEID0_EL0 <bit 0xE> to check support of BR_RETURN_RETIRED and should be 1. 2. Program PMU counter to monitor BR_RETURN_RETIRED. 3. Refer algorithm of BR_IMMED_RETIRED event to generate the BR_RETURN_RETIRED event. For BR_RETIRED 1. Read PMCEID1_EL0 <bit 0x1> to check support of BR_RETIRED and should be 1. 2. Program PMU counter to monitor BR_RETIRED. 3. Refer algorithm of BR_IMMED_RETIRED event to generate the BR_RETIRED event. • For BR_MIS_PRED_RETIRED 1. Read PMCEID1_EL0 <bit 0x2> to check support of R_MIS_PRED_RETIRED & should be 1. 2. Program required counter & PMU. 3. Call stimulus with to generate event. 4. In stimulus, run a conditional (if-else) branch statements with continues true condition & a false condition in between will generate this. 5. Read PMU counter. 6. Repeat steps 3 to 5 with different scaling factors (SMALL, MEDIUM, LARGE) and make sure the counters moved accordingly. For BR_IMMED_PRED_RETIRED 1. Check the BR_IMMED_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor.

Test number	Rule ID	Level	Rule	Algorithm
			0x811E BR_INDNR_RETIRED Branch instruction architecturally executed, indirect excluding procedure return	2. Program PMU counter to monitor BR_IMMED_PRED_RETIRED event. 3. Refer algorithm of BR_IMMED_RETIRED to generate & monitor this event. For BR_IMMED_MIS_PRED_RETIRED 1. Check the BR_IMMED_MIS_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_MIS_PRED_RETIRED to generate and monitor event. For BR_RETURN_PRED_RETIRED 1. Check the BR_RETURN_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_MIS_PRED_RETIRED to generate and monitor event. For BR_RETURN_MIS_PRED_RETIRED 1. Check the BR_RETURN_MIS_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_MIS_PRED_RETIRED to generate and monitor event. For BR_TAKEN_PRED_RETIRED 1. Check the BR_TAKEN_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_MIS_PRED_RETIRED to generate and monitor event. For BR_TAKEN_MIS_PRED_RETIRED 1. Check the BR_TAKEN_MIS_PRED_RETIRED

Test number	Rule ID	Level	Rule	Algorithm
				event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_MIS_PRED_RETIRED to generate and monitor event. For BR_SKIP_PRED_RETIRED 1. Check the BR_SKIP_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_MIS_PRED_RETIRED to generate and monitor event. For BR_SKIP_MIS_PRED_RETIRED 1. Check the BR_SKIP_MIS_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_MIS_PRED_RETIRED to generate and monitor event. For BR_PRED_RETIRED 1. Check the BR_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_MIS_PRED_RETIRED to generate and monitor event. 4. For BR_INDNR_PRED_RETIRED 5. Check the BR_INDNR_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 6. Program required counter & PMU. 7. Refer algorithm of BR_IND_RETIRED to generate and monitor event. For BR_INDNR_MIS_PRED_RETIRED 1. Check the BR_INDNR_MIS_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor.

Test number	Rule ID	Level	Rule	Algorithm
				2. Program required counter & PMU. 3. Refer algorithm of BR_IND_RETIRED to generate and monitor event. For BR_IND_PRED_RETIRED 1. Check the BR_IND_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_IND_RETIRED to generate and monitor event. For BR_IND_MIS_PRED_RETIRED 1. Check the BR_IND_MIS_PRED_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Refer algorithm of BR_IND_RETIRED to generate and monitor event. For BR_IND_RETIRED 1. Check the BR_IND_RETIRED event support & get event number from PAL API should be filled by the vendor. 2. Program required counter & PMU. 3. Call stimulus to generate event. 4. In stimulus, call assembly function with BR instruction multiple times. 5. BR instruction call depends on some conditional instructions and make the condition true and false to generate predicted and mis predicted branch events. 6. Read PMU counter. 7. Repeat steps 3 to 6 with different scaling factors (SMALL, MEDIUM, LARGE) and make sure the counters moved accordingly. For BR_INDNR_RETIRED 1. Check the BR_INDNR_RETIRED event support & get event number from PAL API should be filled by the vendor 2. Program required counter & PMU 3. Refer algorithm of BR_IND_RETIRED to generate and monitor event
PMU Linux test	PMU_EV_09	L7	The BR_RETIRED event must count unconditional taken branches.	Refer to PMU_EV_08 rule, BR_RETIRED event algorithm.

Test number	Rule ID	Level	Rule	Algorithm
PMU Linux test	PMU_EV_10	L7	The events listed in the following table are used to measure Section B.3.6 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	For MEM_ACCESS Refer PMU_EV_02, MEM_ACCESS algorithm. For BUS_ACCESS 1. Read PMCEID0_EL0 <bit 0x19> to check support of BUS_ACCESS event and should be 1 2. Program PMU counter to monitor BUS_ACCESS event. 3. Refer to MEM_ACCESS algorithm to generate and monitor event For BUS_ACCESS_RD 1. Check BUS_ACCESS_RD event support & get event number from PAL API should be filled by the vendor. 2. Program PMU counter to monitor BUS_ACCESS_RD event. 3. Refer to MEM_ACCESS algorithm to generate and monitor event. For BUS_ACCESS_WR 1. Check BUS_ACCESS_WR event support & get event number from PAL API should be filled by the vendor. 2. Program PMU counter to monitor BUS_ACCESS_WR event. 3. Refer to MEM_ACCESS algorithm to generate and monitor event. For MEM_ACCESS_RD 1. Check BUS_ACCESS_RD event support & get event number from PAL API should be filled by the vendor. 2. Program PMU counter to monitor BUS_ACCESS_RD event. 3. Refer to MEM_ACCESS algorithm to generate and monitor event. For MEM_ACCESS_WR 1. Check MEM_ACCESS_WR event support & get event number from PAL API should be filled by the vendor. 2. Program PMU counter to monitor MEM_ACCESS_WR event. 3. Refer to MEM_ACCESS algorithm to generate and monitor event

Test number	Rule ID	Level	Rule	Algorithm
1103	PMU_EV_11	L7	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.	Read ID_AA64DFR0_EL1.MTPMU[bits [51:48]] and should be either 0xF or 0x1.
1104	PMU_BM_1	L7	The base server system must implement bandwidth monitors for: Memory interface	1. Get the number of memory ranges from SRAT ACPI table. 2. Get the corresponding proximity domain for the memory range. 3. Get the PMU node associated with memory controller and obtained proximity domain from APMT ACPI table. 4. Get base address of the PMU node from APMT table. 5. Confirm the PMU supports at least 3 counters for 3 bandwidth events (Inbound READ, Inbound Write, Inbound Total) 6. Get base address of proximity domain from SRAT ACPI table. 7. Program PMU_EVTYPERn to monitor a specific event. The implementation defined event IDs will be obtained from PAL API. 8. Configure PMCR.E to enable PMU 9. Configure PMU_CNTENSET<n> to enable counter n 10. Allocate 4MB of memory at base address obtained for the corresponding proximity domain. 11. Initiate read/write to memory for 2 MB and read PMU_EVCNTRn for first count. 12. Configure PMU_CNTENCLR<n> to disable counter n. 13. Configure PMU_CNTENSET<n> to enable counter n. 14. Initiate read/write to memory for 4 MB and read PMU_EVCNTRn for second count. 15. Configure PMCR.E to disable PMU 16. The bandwidth measurements of second run should be greater than first run.

Test number	Rule ID	Level	Rule	Algorithm
1107	PMU_BM_2	L7	The base server system must implement bandwidth monitors for: PCIe interface	1. Get the number of ECAM regions in the system. 2. Get the PMU node associated with PCIE Root complex from APMT ACPI table. 3. Get base address of the PMU node from APMT table. 4. Confirm the PMU supports at least 10 counters (6 counters for bandwidth events, 4 for latency events) 5. Program PMU_EVTYPERn to monitor a specific event. The implementation defined event IDs will be obtained from PAL API. 6. Configure PMCR.E to enable PMU 7. Configure PMU_CNTENSET<n> to enable counter n 8. Generate PCIe workload through config reads/writes for device number 0-10 9. Configure PMU_CNTENCLR<n> to disable counter n 10. Read PMU_EVCNTRn for current count (delta into consideration for the results) 11. Configure PMCR.E to disable PMU 12. Repeat steps 6-10 , use device 0-20 for generating workload. 13. Bandwidth measurement for second run must be greater than first run 14. Latency monitor should have a valid count value as per workload generated.
	PMU_BM_3	L7	The base server system must implement bandwidth monitors for: External accelerator interface.	Not Implemented Generating traffic for external accelerator for monitoring bandwidth events is IMP DEF.
	PMU_BM_4	L7	The base server system must implement bandwidth monitors for Chip-to-chip interface.	Not Implemented Generating traffic for chip2chip for monitoring bandwidth events is IMP DEF.
1105	PMU_MEM_1	L7	The base server system must implement average latency monitors for each memory interface.	1. Get the number of memory ranges from SRAT ACPI table. 2. Get the corresponding proximity domain for the memory range, 3. Get the PMU node associated with memory controller for the obtained proximity domain from APMT ACPI table. 4. Get base address of proximity domain from SRAT ACPI table.

Test number	Rule ID	Level	Rule	Algorithm
				5. Get base address of the PMU node from APMT table. 6. Confirm the PMU supports at least 2 counters (2 counters for latency). 7. Program PMU_EVTYPERn to monitor a specific event. The implementation defined event IDs will be obtained from PAL API. 8. Configure PMCR.E to enable PMU. 9. Configure PMU_CNTENSET to enable counter n. 10. Allocate memory at base address obtained for the corresponding proximity domain. 11. Initiate read/write to memory and very the scale. 12. Configure PMU_CNTENCLR to disable counter n. 13. Read PMU_EVCNTRn for current count (delta into consideration for the results) 14. Configure PMCR.E to disable PMU.
	PMU_SYS_1	L7	Each monitor for an interface must be capable of measuring all of the following measurements simultaneously: • If the interface supports outbound read traffic, average outbound read latency. • If the interface supports outbound traffic, total outbound bandwidth. • If the interface supports inbound read traffic, average inbound read latency. • If the interface supports inbound traffic, total inbound bandwidth. That is, up to 4 measurements (typically requiring up to 6 event counters) simultaneously.	Covered as a part of tests 1104, 1105, 1107.
	PMU_SYS_2	L7	Each monitor for an interface must be capable of measuring all of the following measurements simultaneously: • If the interface supports outbound read traffic, outbound read bandwidth and average outbound read latency. • If the interface supports outbound write traffic, outbound write bandwidth. • If the interface supports inbound read traffic, inbound read bandwidth and average inbound read latency. • If the interface supports inbound write traffic, inbound write bandwidth.	Covered as a part of tests 1104, 1105, 1107

Test number	Rule ID	Level	Rule	Algorithm
			That is, up to 6 measurements (typically requiring up to 8 event counters) simultaneously.	
1108	PMU_SYS_5	L7	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.	1. Get the number of memory ranges from SRAT ACPI table. Test fails if no memory ranges found. 2. Get Local PE info such as PE UID form PE info table, PE proximity domain form SRAT info table. 3. Get the PMU node associated with memory controller from APMT table local to the primary PE 4. Get base address of the PMU node from APMT table. 5. Confirm the PMU supports at least 3 counters for 3 bandwidth events (Local, Remote and All traffic) 6. Program PMU_EVTYPERn to monitor a specific event. The implementation defined event IDs will be obtained from PAL API. 7. Configure PMCR.E to enable PMU 8. Configure PMU_CNTENSET<n> to enable counter n 9. Get remote PE info. Remote PE proximity domain and UID from SRAT info table and Remote PE index form PE info table. 10. Get Base address of the memory controller local to the PE from SRAT info table. 11. Allocate 4MB of memory at base address obtained for the corresponding proximity domain. 12. Generate memory traffic to the local memory controller from both local PE and remote PE. 13. Initiate read/write to memory for 2 MB and read PMU_EVCNTRn for first count. 14. Configure PMU_CNTENCLR<n> to disable counter n. 15. Configure PMU_CNTENSET<n> to enable counter n. 16. Initiate read/write to memory for 4 MB and read PMU_EVCNTRn for second count. 17. Configure PMCR.E to disable PMU The bandwidth measurements of second run should be greater than first run.
1109	PMU_SYS_6	L7	For interfaces that carry multiple types of traffic, each monitor must be capable of	

Test number	Rule ID	Level	Rule	Algorithm
			filtering monitored traffic that is based on its traffic type.	
	PMU_SYS_7	L7	Each significant cache in the base server system must be capable of measuring cache effectiveness	Will be covered as part of PMU_EV_02
	PMU_SEC_1	L7	When deployed in production systems, performance monitors must not expose Secure data to untrusted software.	Not Implemented No generic secure PMU events are mentioned in the specification. Also don't have access to secure memory map from UEFI shell.

2.11 MPAM

Test number	Rule ID	Level	Rule	Algorithm
1001	S_L7MP_01	L7	PEs must implement the MPAM extension. See FEAT_MPAM in [2].	Read the system register ID_AA64PFR1_EL1 Check for the below condition if ((ID_AA64PFR1_EL1.MPAM > 0) || (ID_AA64PFR1_EL1.MPAM_frac > 0))
1001	S_L7MP_02	L7	PEs must implement a minimum of 16 physical partition IDs.	Read the system register MPAMIDR_EL1 Check for the below condition if(MPAMIDR_EL1.PARTID_MAX > 16) ?
1002	S_L7MP_03	L7	The implementation must provide MPAM Cache Storage Usage monitors for the last-level cache.	1. Check if PE implements MPAM 2. Get Last level Cache index from PPTT ACPI table 3. Get Cache ID corresponding to the Last level cache index 4. Get MSC count from MPAM ACPI Table. Test fails if no nodes found, 5. Iterate though all the MSC nodes and visit each node which is of type PE Cache and has Cache ID matching the Last level cache. 6. Check if Last level cache supports CSU monitor. MPAMF_MSMON_IDR.MSMON_CSU should read 0b1

Test number	Rule ID	Level	Rule	Algorithm
1002	S_L7MP_04	L7	Last-level cache must provide a minimum of 16 Cache Storage Usage monitors	1. Follow Steps 1 - 5 from the rule S_L7MP_03. 2. Check if Last level cache has at least 16 CSU monitors. MPAMF_CSUMON_IDR.IDR_NUM_MON >=16
1003	S_L7MP_05	L7	The implementation must provide MPAM Memory Bandwidth Usage monitors (MBWUs) for the interfaces that provide general purpose memory	1. Check if PE Implements MPAM 2. Get MSC count from MPAM ACPI Table. Test fails if no nodes found. 3. Confirm MSC mpam version is 1.1 using MPAM_AIDR register 4. Read MPAM_EL2 register and store it in a temporary variable. 5. Program MPAM_EL2 with default PARTID and PMG values to generate traffic, 6. Visit each MPAM MSC node of type Memory and check for following conditions. 6. Check if MSC supports MBWU monitor. MPAMF_MSMON_IDR.MSMON_MBWU should read 0b1 8. Select Resource Instance if RIS feature is implemented 9. Get Memory Base address and length from SRAT ACPI table. 10. Configure MBWU monitors and enable monitoring 11. Perform a memory copy of 64KB to the address obtained from SRAT table. 12. Measure the MPAM MBWU monitor for bandwidth count. 13. The monitor should count both read and write bandwidth, the bandwidth count should be twice of the buffer size of 64KB. 14. Free the Allocated buffers.
	S_L7MP_06	L7	The MBWUs must implement the MPAM v1.1 64-bit MBWU extension. See FEAT_MPAMv1p1 in [2].	Covered by SBSA test 1003

Test number	Rule ID	Level	Rule	Algorithm
1004	S_L7MP_07	L7	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	1. Check if PE implements MPAM v1.1 2. Get MSC count from MPAM ACPI Table. Test fails if no nodes found. 3. Visit each MPAM MSC node of type Memory and check for following conditions. 4. Check if MSC supports MBWU monitor. MPAMF_MSMON_IDR.MSMON_MBWU should read 0b1 5. Check if MSC supports MPAM Long MBWU monitor, MPAMF_MSMON_IDR.HAS_LONG should read 0b1 6. Get Max bandwidth details from HMAT ACPI table. 6. Check if 63-bit counter is implemented MPAMF_MBWUMON_IDR.LWD == 1) and Max bandwidth supported is 1.6TB.
1005	S_L7MP_08	L7	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. Get MSC count from MPAM ACPI Table. Test fails if no nodes found 2. Get Number of USB, UART and SATA controllers and their base addresses form Peripheral info table. 3. Visit Each MSC node and check if the base address overlaps with adjacent MSC nodes and USB , UART and SATA controllers

Test number	Rule ID	Level	Rule	Algorithm
1006	S_L7MP_03	L7	The implementation must provide MPAM Cache Storage Usage monitors for the last-level cache. Last-level cache must provide a minimum of 16 Cache Storage Usage monitors	1. Get Cache size, Max PARTID, Max PMG, CSUMON count, CPOR NODES 2. Configure CPOR with Max PARTID and 75% of Cache size 3. Allocate Source and Destination buffers for 50% of Cache size 4. Configure monitor with PMG1 and Max PARTID, 5. Enable CSU Monitor. 6. Generate PE traffic with PMG2 and Max PARTID 7. Read CSU Monitor and store in storage_value1 8. Disable and Reset Monitor. 9. Enable CSU monitor. 10. Generate PE traffic with PMG1 and Max PARTID 11. Read CSU Monitor and store in storage_value2. 12. Disable and Reset Monitor. 13. Test fails if storage_value1 is nonzero or storage_value2 is zero 14. Restore MPAM_EL2 and free the buffers.

Appendix A Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1 Issue 02

Change	Location
First release	-

Table A-2 Difference between Issue 02 to Issue 03

Change	Location
New PCIe tests are added.	See **PCIE**

Table A-3 Difference between Issue 03 to Issue 04

Change	Location
Added SBSA Level 6 PE and SMMU tests.	See • **PE** • **IO Virtualization**
Added new PCIe tests and Exerciser tests that are related to Address Translation Service, Peer-to-Peer, and ACS rules.	See **PCIE**

Table A-4 Difference between Issue 04 to Issue 05

Change	Location
Memory test for unpopulated address space access waived off and implemented for bare-metal.	See **Test Scenarios**
Fixes in test PE and GIC.	See • **PE** • **GIC**
Enabling mmio prints dumps on demand.	See **Test Scenarios**
Enabled new ARI test	See **Test Scenarios**
Enabled bare-metal driver for GIC.	See **GIC**
PCIe Enumeration enhancements.	See **PCIE**
Bug fix and enhancements related to P2P, SMMU and PCIe.	See • **IO Virtualization** • **PCIE**
Updated interrupt related test cases.	See **Test Scenarios**
Additional support provided for running ACS on bare-metal.	See **Test Scenarios**

Table A-5 Difference between Issue 05 to Issue 06

Change	Location
No technical changes	

Table A-6 Difference between Issue 06 to Issue 0701-01

Change	Location
Added SBSA rule id mapping to test	See **Test Scenarios**
New RAS section added	See **RAS**
New PMU section added	See **PMU**
New MPAM section added	See **MPAM**

Table A-7 Difference between Issue 0701-01 to Issue 0701-02

Change	Location
Added SBSA rule id mapping to test	See **Test Scenarios**
Memory Map Test base number changed from 1300 to 100 series	
PCIe Test base number changed from 400 to 800 series	
GIC Test base number changed from 100 to 200 series	
SMMU Test base number changed from 700 to 300 series	
Watchdog Test base number changed from 300 to 700 series	
PCIe exerciser Test base number changed from 800 to 900 series	

Table A-7 Difference between Issue 0701-02 to Issue 0701-03

Change	Location
Added PCIe rules	See **PCIe**
Change in numbering of exerciser tests	See **PCIe**