



Architecture Compliance Test Scenario ES

Revision: r0p9

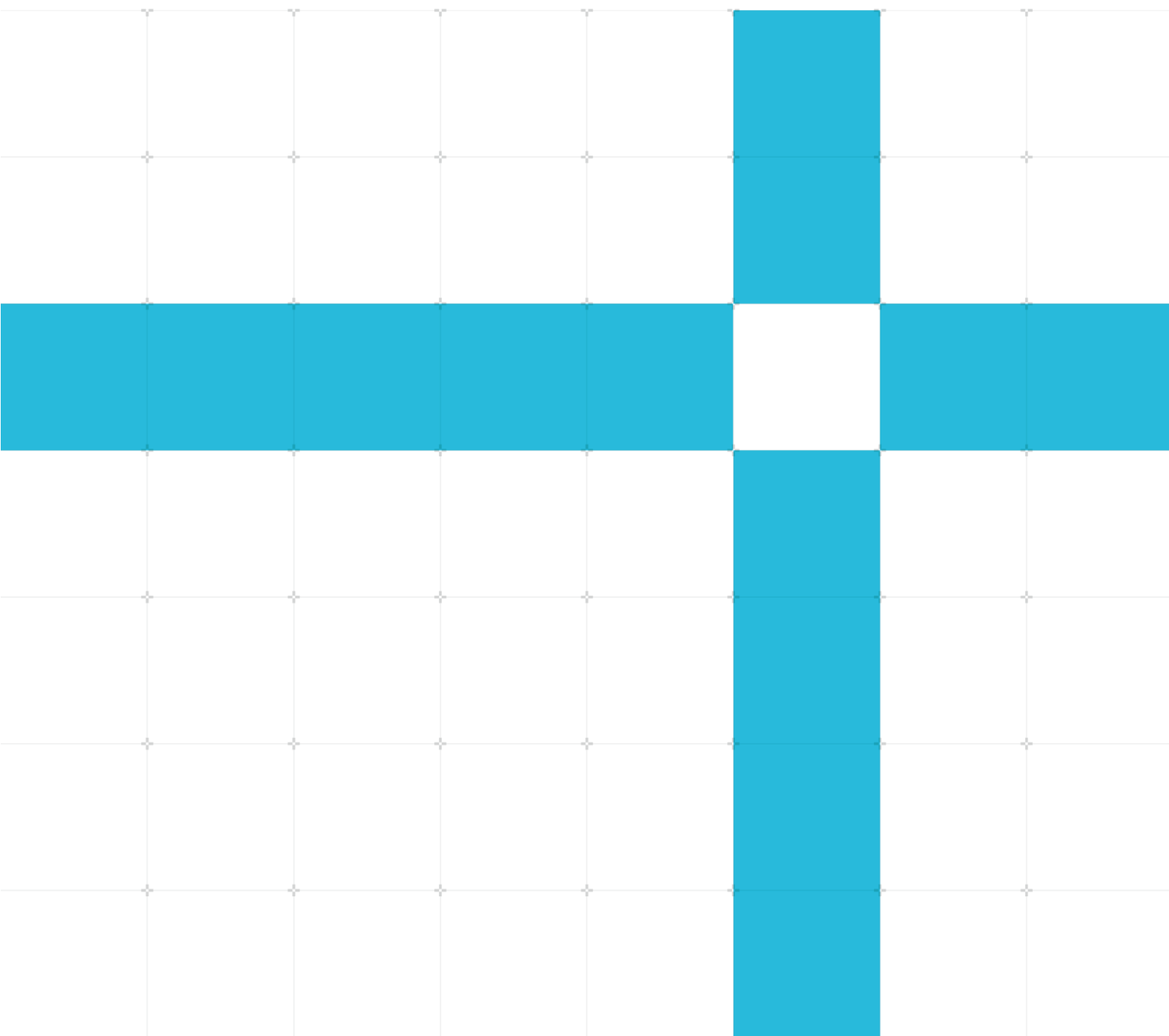
Arm Base System Architecture

Non-Confidential

Copyright © 2021 Arm Limited (or its affiliates).
All rights reserved.

Issue 01

PJDOC-2042731200-7090



Arm Base System Architecture

Architecture Scenario

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

| Issue | Date | Confidentiality | Change |
|-------|--------------|------------------|------------------|
| 01 | 26 July 2021 | Non-Confidential | Beta release 0.9 |

Non-Confidential Proprietary Notice

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

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

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

This document may include technical inaccuracies or typographical errors.

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

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

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

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

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

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

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 an Alpha product, that is a product under development.

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..... | 5 |
| 1.1 Product revision status..... | 5 |
| 1.2 Intended audience..... | 5 |
| 1.3 Conventions..... | 5 |
| 1.3.1 Glossary..... | 5 |
| 1.3.2 Typographical Conventions..... | 6 |
| 1.4 Additional Reading..... | 6 |
| 1.5 Feedback..... | 7 |
| 1.5.1 Feedback on this product..... | 7 |
| 1.5.2 Feedback on content..... | 7 |
| 2 Base System Architecture..... | 8 |
| 2.1 BSA ACS..... | 8 |
| 2.2 PE..... | 9 |
| 2.3 Memory Map..... | 12 |
| 2.4 GIC..... | 13 |
| 2.5 SMMU..... | 15 |
| 2.6 Clock and Timer..... | 20 |
| 2.7 Wakeup Semantics..... | 22 |
| 2.8 Power State Semantics..... | 23 |
| 2.9 Watchdog..... | 23 |
| 2.10 Peripherals..... | 24 |
| 2.11 PCIe..... | 26 |
| 2.12 DeviceID Generation and ITS Groups..... | 44 |
| Appendix A Revisions..... | 48 |

1 Introduction

1.1 Product revision status

The *rm**pn* identifier indicates the revision status of the product described in this book, for example, *r1p2*, where:

rm Identifies the major revision of the product, for example, *r1*.

pn Identifies the minor revision or modification status of the product, for example, *p2*.

1.2 Intended audience

This document is for engineers who are verifying an implementation of Arm® Base System Architecture 1.0.

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 |
|----------------------------|---|
| <i>italic</i> | 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 <u>underline</u> | 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: <pre>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></pre> |
| 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 Additional Reading

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

Table 1-1 Arm publications

| Document name | Document ID | Licensee only |
|--|---------------|---------------|
| Arm® Base System Architecture 1.0 | DEN0094A | No |
| Arm® Base Boot Requirements 1.0 | DEN0107A | No |
| Arm® Architecture Reference Manual Armv8, for Armv8-A Architecture | ARM DDI 0487F | No |

Table 1-2 Other publications

| Document ID | Document name |
|-------------|--|
| None | PCI Express Base Specification Revision 5.0, Version 1.0 |

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-accs@arm.com and give:

- The title Arm Base System Architecture Scenario.
- The number PJDOC-2042731200-6896.
- 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 Base System Architecture

Base System Architecture (BSA) specifies a hardware system architecture based on Arm 64-bit architecture, that system software like operating systems, hypervisors, and firmware can rely on. The document addresses PE features and key aspects of system architecture. The primary goal of the document is to ensure sufficient standard system architecture to enable a suitably built single OS image to run on all the hardware compliant with the specifications.

Arm does not mandate compliance with this specification. However, Arm anticipates that OEMs, ODMs, cloud service providers, and software providers will require compliance to maximize Out-of-Box software compatibility and reliability.

2.1 BSA ACS

BSA ACS provides tests for validating the compliance of a platform with BSA specifications. 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 a hierarchy is consistent with the target hardware subsystem which is validated by a test. A test may check for different parameters of the hardware subsystem.

The tests are classified as:

- PE
- Memory Map
- GIC
- SMMU
- Clock and Timer
- Wakeup Semantics
- Power State Semantics
- Watchdog
- Peripherals
- PCIe
- DeviceID Generation and ITS Groups

2.2 PE

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|---------|---|---|
| 1 | B_PE_01 | All PEs are architecturally symmetric except for the permitted exceptions listed in Section A of BSA specification. | CPU System register read. Read all the processor ID registers from all PEs and then compare values with the main PE. |
| 2 | B_PE_02 | The number of PEs must not exceed: <ul style="list-style-type: none"> Eight, when the interrupt controller is compliant to GICv2. 2²⁸ when the interrupt controller is compliant to GICv3 or higher. | Read from ACPI MADT table. |
| 3 | B_PE_03 | PEs must implement the Advanced SIMD and FEAT_FP extensions. | CPU System register read: ID_AA64PFR0_EL1 must indicate support bits [23:20]. |
| 4 | B_PE_04 | PEs must support 4KB translation granules at stage 1. | ID_AA64MMFR0_EL1 must indicate support for 4KB and 64KB granules for all cores. |
| 5 | B_PE_05 | All PEs are coherent and in the same Inner Shareable domain. | ID_MMFR0_EL1.InnerShr must indicate hardware coherency support for InnerShr across all cores, ShreLvl must be 0001 across all cores (later is mandated for Armv8). Functional verification is optional. |
| 6 | B_PE_06 | Where export restrictions allow, PEs must implement cryptography extension support for FEAT_AES, FEAT_SHA1 and FEAT_SHA256. | CPU System register read: Bits [4:15] of ID_AA64ISAR0_EL1 must be supported |
| 7 | B_PE_07 | PEs must implement little-endian support. | CPU System register read: SCTLR.EE should be supported. |
| 8 | B_PE_08 | PEs must implement EL1 and ELO in the AArch64 Execution state. | ID_AA64PFR0_EL1.ELO and ID_AA64PFR0_EL1.EL1 should be supported. |
| 9 | B_PE_09 | PEs must implement the FEAT_PMU extension, and the base system must expose a minimum of four programmable PMU counters to the operating system. | If ID_AA64DFR0_EL1.PMUVer is supported, then PMCR_ELO.N should be greater than three. |
| 10 | B_PE_10 | The PMU overflow signal from each PE must be wired to a unique PPI interrupt with no intervening logic. | If ID_AA64DFR0_EL1.PMUVer is supported, Install ISR and verify PMU Overflow Interrupt by programming System register. |
| 11 | B_PE_11 | Each PE must implement a minimum of six breakpoints, two of which must match virtual address, contextID or VMID. | 1. Read ID_AA64DFR0_EL1 and check number of breakpoints. 2. Read DBGBCR<n>_EL1 & check type of breakpoint. |
| 12 | B_PE_12 | Each PE implements a minimum of four synchronous watchpoints. | ID_AA64DFR0_EL1.WRPs must be greater than 2. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|---------|--|--|
| 13 | B_PE_13 | PEs must implement the FEAT_CRC32 instructions. | ID_AA64ISAR0_EL1.CRC32 should be supported. |
| | B_PE_14 | Implementation of SVE or SVE2 is optional. If implemented, Arm strongly recommends that the performance of well-optimized SVE or SVE2 code is no worse than code which uses the equivalent NEON instructions. | Not Implemented Requires the performance comparison between SVE/SVE2 and NEON code. This is not feasible. |
| 14 | B_PE_15 | <p>If FEAT_PAuth (Pointer Authentication), mandatory from Armv8.3, is implemented, Arm recommends that the standard algorithm defined by the Arm architecture is implemented for address and generic authentication.</p> <ul style="list-style-type: none"> If an alternative algorithm is used, it must be at least as cryptographically strong as the Arm recommended algorithm. | If pointer signing is implemented, check ID_AA64ISAR1_EL1[11:4] for address authentication and ID_AA64ISAR1_EL1[31:24] for generic authentication. |
| - | B_PE_16 | <p>If FEAT_MTE (Memory Tagging Extension), is implemented,</p> <ul style="list-style-type: none"> The implementation can be full including instructions and checks or 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 indicate the memory ranges to the OS. | Not implemented in the current release. |
| 15 | B_PE_17 | 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. |
| 51 | B_PE_18 | PEs must implement Non-secure EL2 in AArch64. | ID_AA64PFR0_EL1.EL2 must be supported. |
| 52 | B_PE_19 | PEs must support 4KB translation granules at stage 2. | ID_AA64MMFR0_EL1.TGran4_2 must be supported. |
| 53 | B_PE_20 | The translation granules supported at stage 2 must match those supported at stage 1. | Check AA64MMFR0_EL1 register. |
| 54 | B_PE_21 | The base system must expose a minimum of two programmable PMU counters to a hypervisor. | Check whether minimum two programmable PMU counters are exposed. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|--|
| 55 | B_PE_22 | Two of the implemented breakpoints in each PE must be able to match on VMID. See B_PE_11. | Read DBGBCR<n>_EL1 & check type of breakpoint. |
| 76 | B_PE_23 | PEs must implement EL3 in the AArch64 Execution state. | ID_AA64PFR0_EL1.EL3 must be supported. |
| 76 | B_PE_24 | PEs must implement Secure state. | ID_AA64PFR0_EL1.EL3 must be supported. |
| 16 | B_SEC_01 | PEs must implement the restrictions on speculation that are introduced in the Arm v8.5 extensions to the Arm architecture [2] and SCXTNUM_ELx registers as indicated by ID_AA64PFR0_EL1.CSV2==b0010 and ID_AA64PFR0_EL1.CSV3==b0001. See FEAT_CSV2 and FEAT_CSV3 in [2]. | ID_AA64PFR0_EL1.CSV2 and ID_AA64PFR0_EL1.CSV3 must be supported. |
| 17 | B_SEC_02 | PEs implement the PSTATE/CPSR SSBS (Speculative Store Bypass Safe) bit and the instructions to manipulate it. See FEAT_SSBS in [2]. This is identified by ID_AA64PFR1_EL1.SSBS==b0010. | ID_AA64PFR1_EL1.SSBS must be supported. |
| 18 | B_SEC_03 | PEs implement the CSDB, SSBB, and PSSBB barriers. | ID_AA64PFR1_EL1.SSBS must be supported. |
| 19 | B_SEC_04 | PEs implement the FEAT_SB. This is indicated by ID_AA64ISAR1_EL1.SB== b0001. | ID_AA64ISAR1_EL1.SB must be supported. |
| 20 | B_SEC_05 | 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 FEAT_SPECRES in [2]. Support for the instructions is indicated by ID_AA64ISAR1_EL1.SPECRES==b0001. | ID_AA64ISAR1_EL1.SPECRES must be supported. |

2.3 Memory Map

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|--|--|
| 102 | B_MEM_01 | All memory accesses, whether they access memory space that is populated or not, must respond within finite time, to avoid the possibility of system deadlock. | <ol style="list-style-type: none"> 1. Initialize the exception handlers. 2. Get the address range from the system table or memory map. 3. Try to access the memory. 4. Exception may or may not occur, but the test should continue to proceed. 5. Once all memory ranges are accessed, the test is considered as pass. |
| - | B_MEM_02 | Where a memory access is to an unpopulated part of the addressable memory space, accesses must be terminated in a manner that is presented to the PE as either a precise Data Abort, or as a system error interrupt, or an SPI, or LPI interrupt to be delivered to the GIC. | Not implemented in the current release. |
| 104 | B_MEM_03 | All Non-secure on-chip DMA requestors in a base system that are expected to be under the control of the operating system or hypervisor must be capable of addressing all the Non-secure address space. | <ol style="list-style-type: none"> 1. Check if the PCIe device is capable of 64-bit DMA. 2. If not, check if it is present behind an SMMU |
| - | B_MEM_04 | If the Requester goes through an SMMU, then the Requester must be capable of addressing all the Non-secure address space when the SMMU is turned off. | Covered by Test 104 |
| 103 | B_MEM_05 | All PEs must be able to access all the Non-secure address space. | <ol style="list-style-type: none"> 1. For the current PE, read ID_AA64MMFR0_EL1[3:0] to get max number of bits that PE can access. 2. Check if the maximum accessible memory is accessible by PE. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|---|
| - | B_MEM_06 | Non-secure off-chip devices that cannot directly address all the Non-secure address space must be placed behind a stage 1 SMMU that is compatible with the Arm SMMUv2 or SMMUv3 specification, that has an output address size large enough to address all the Non-secure address space. (See Section 3.7 [3]). | Covered by Test 104. |
| - | B_MEM_07 | It is possible for the progress of a memory transaction to depend on a second memory access, the system must avoid deadlock if the memory access gets ordered behind the original transaction. | Not Implemented. The sequence to generate deadlock is not repeatable in an environment such as UEFI shell. |
| - | B_MEM_08 | The system must provide some memory that is mapped in the Secure address space. | Not Implemented. Does not have access to Secure memory map from UEFI shell. |
| - | B_MEM_09 | This Secure memory must not be aliased in the Non-secure address space. | Not Implemented. Does not have access to secure memory map from UEFI shell. |

2.4 GIC

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|--|
| 201 | B_GIC_01 | A base system must present operating systems and hypervisors with the interfaces defined by one of the following: <ul style="list-style-type: none"> Generic Interrupt Controller (GIC) v2 interrupt controller. GICv2 interrupt controller with GICv2m extension. GICv3 interrupt controller. | Check for the version from the platform information. If it is not present in platform information, then read GICD_PIDR2. |
| 202 | B_GIC_02 | Limitations and valid configurations are allowed in a base system. | Check that PCIe support should not be present if GICv2. |
| 203 | B_GIC_03 | If the system includes PCI Express and GICv3 interrupt controller is supported, then the GICv3 interrupt controller must implement ITS and LPI. | Check for the GIC version and PCIe presence in the system, and check for ITS or LPI if present. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-------------|---|---|
| 204 | B_GIC_04 | If a GICv3 interrupt controller is supported, then the interrupt controller must support two security states. | Read security implemented from GICD_CTLR.DS bit. |
| 205 | B_GIC_05 | The system may implement at least eight Non-secure SGIs assigned to interrupt IDs 0-7. | <ol style="list-style-type: none"> 1. Check whether the distributor forwards Non-secure Group 1 interrupts. 2. If NS Group 1 interrupts are forwarded, try to enable SGI INTID 0-15 by writing to the distributor or redistributor enable register. 3. Check which interrupt IDs are enabled afterwards; Secure interrupts return 0, and Non-secure interrupts return 1. |
| 206 | B_PPI_01 | Check PPI assignments for Operating System. | <ol style="list-style-type: none"> 1. Detect the interrupt IDs. 2. Install ISRs for those IDs. 3. Trigger the conditions to assert the interrupts. 4. Check whether the interrupt has been received. |
| 226 | B_PPI_02 | Check PPI assignments for Hypervisor. | <ol style="list-style-type: none"> 1. Detect the interrupt IDs. 2. Install ISRs for those IDs. 3. Trigger the conditions to assert the interrupts. 4. Check whether the interrupt has been received. |
| - | B_PPI_03 | Check PPI assignments for Platform Security. | <p>Not Implemented.</p> <p>BSA ACS executes from Non-secure mode. Cannot access Secure PPI interrupts.</p> |
| 251 | Section I.6 | Check SPIs which are assigned to MSIs are edge-triggered. | <ol style="list-style-type: none"> 1. Get the number of MSI frames. 2. Loop through number of SPIs for that frame. 3. Check by reading the value of GICD_ICFGRn register for SPI. |
| 252 | Section I.9 | Check GICv2m MSI Frame Register Configuration. | <ol style="list-style-type: none"> 1. Get the number of MSI frames. 2. Loop through number of SPIs for that frame. 3. Check MSI_TYPER is RO. 4. Check for SPI_ID is 32-1020. 5. Check MSI_IIDR is RO. |
| 253 | Section I.6 | Check GICv2m MSI to SPI generation functional test. | <ol style="list-style-type: none"> 1. Get the number of MSI frames. 2. For each frame get an SPI_ID. 3. Install a Handler. 4. Trigger an SPI using MSI_SETSPI_NS register. 5. Check if interrupt is triggered. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-------------|--|---|
| 254 | Section I.5 | SPIs that are allocated to MSIs must only be controllable by the GICv2m MSI registers. | <ol style="list-style-type: none"> 1. Get the number of MSI frames. 2. For each frame get an SPI_ID. 3. Install a Handler. 4. Generate SPI using GICD_ISPENDR register. 5. No Interrupt should be generated. 6. Trigger an SPI using the MSI_SETSPI_NS register. 7. Check the Interrupt is triggered as SPIs should be controlled by the MSI frame register instead of other GICD Registers. |

2.5 SMMU

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|---|
| 301 | B_SMMU_01 | All the System MMUs presented to an OS or hypervisor must be compliant with the same architecture version. | For all SMMU controllers in the system, check if they are based on either the v2 or v3 architecture. |
| 302 | B_SMMU_02 | The SMMU must support the translation granule sizes that are supported by the PEs. | <ol style="list-style-type: none"> 1. Read out the granule size support for the PE (AA64MMFR0_EL1.TGran_x). 2. Verify whether this is congruent to the granule sizes supported by the SMMU. |
| 303 | B_SMMU_03 | If PEs implement FEAT_LVA (ID_AA64MMFR2_EL1.VARange = 0b0001), the SMMU must support extended virtual addresses (SMMU_IDR5.VAX = 0b01). | <ol style="list-style-type: none"> 1. Read out: ID_AA64MMFR2_EL1.VARange. 2. If enabled, check whether SMMU_IDR5.VAX is set. |
| 304 | B_SMMU_04 | If PEs use Armv8.4 and can issue TLB range invalidation instructions, the SMMU must support range invalidation. | <ol style="list-style-type: none"> 1. Read out: ID_AA64ISAR0_EL1.TLB. 2. If enabled, check whether SMMU_IDR3.RIL is set. |
| - | B_SMMU_05 | All DVM receivers visible to the Normal world software in a system must receive all DVM messages initiated by a DVM requester. This would require that the DVM capabilities of the SMMU and the interconnect are the same or a superset of the initiator (typically PE). | Covered by Test 304. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|---|
| 305 | B_SMMU_06 | This means that, if PEs implement FEAT_LPA (ID_AA64MMFR0_EL1.PARange = 0b0110), then the SMMU must support a 52-bit output size (SMMU_IDR5.OAS = 0b0110). | <p>1. Read out: ID_AA64MMFR0_EL1.PARange.</p> <p>2. If enabled, check whether SMMU_IDR5.OAS equals "0b0110".</p> |
| - | B_SMMU_07 | Devices that operate across non-contiguously allocated memory require stage 1 System MMU functionality. | <p>Not Implemented.</p> <p>This requires device specific drivers to be available and device specific knowledge. This is not available from UEFI shell application.</p> |
| 306 | B_SMMU_08 | <p>If Secure-EL2 is not implemented, stage 1 System MMU functionality that is made visible to an operating system must present the interface of a System MMU compatible with one of the following:</p> <ul style="list-style-type: none"> • The SMMUv2 specification, where each context bank must present a unique physical interrupt to the GIC. • The Arm SMMUv3 specification or higher, where the integration of the System MMUs is compliant with the requirements in Section D [3] | <p>1. Read out: ID_AA64PFR0_EL1.SEL2.</p> <p>2. If enabled, and SMMU handles stage 1 policing, check whether SMMU_AIDR.ArchMajorRev ≥ 3 and SMMU_AIDR.ArchMinorRev ≥ 2.</p> <p>Else, check whether SMMU major revision is at least 2.</p> |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|---|
| - | B_SMMU_09 | <p>If Secure-EL2 is implemented, stage 1 System MMU functionality that is made visible to an operating system must present the interface of a System MMU compatible with the Arm SMMUv3.2, or higher, architecture revision where:</p> <ul style="list-style-type: none"> • The integration of the System MMUs is compliant with the rules in SMMUv3 integration, Section D [3]. • SMMU implementations must provide level 1 or level 2 support for page table resizing. Arm recommends the SMMU implements level 2. If the SMMU implementation provides level 2, then Arm recommends that the PE also provides level 2. | Covered by Test-306. |
| 309 | B_SMMU_11 | MPAM architecture requires that all requesters that can access an MPAM controlled resource must support passing MPAM ID information. Therefore, a SMMUv3.2, or higher, implementation must support the MPAM extension if the requests it serves access MPAM-controlled resources. | <ol style="list-style-type: none"> 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 != 0). |
| - | B_SMMU_12 | All addresses output from a device to an SMMU must lie in a continuous space with no holes. All addresses in this space will be treated equally by the SMMU. There should be no areas within the address space that receive exceptional treatment like bypassing the SMMU. | <p>Not implemented.</p> <p>We require device drivers to generate DMA and a configurable memory map to access the whole address space. Not feasible to generate these transactions from UEFI shell.</p> |
| 307 | B_SMMU_13 | If PE supports 16-bit ASID, The SMMU must implement support for 16-bit ASID. | <ol style="list-style-type: none"> 1. Read out: ID_AA64MMFR0_EL1.ASIDBits. 2. If enabled, and SMMU handles stage 1 policing, check whether SMMU_IDR0.ASID16 is set. |
| 308 | B_SMMU_14 | The SMMU supports little-endian for translation table walks, and at a minimum must match the endianness support of the PEs. | <ol style="list-style-type: none"> 1. Read out: SCTLR_EL1.EE. 2. If enabled, check whether SMMU_IDR0.TTENDIAN is congruent with this value. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|--|
| 351 | B_SMMU_16 | If a device is assigned and passed through to an operating system under a hypervisor, then the memory transactions of the device must be subject to stage 2 translation, allocation of memory attributes, and application of permission checks, under the control of the hypervisor. | Depending on the architecture revision of the SMMU, check whether SMMU_IDR0.S2TS (v2) or SMMU_IDR0.S2P (v3) is enabled. |
| - | B_SMMU_17 | From a hardware perspective, this means that a base system supporting a protection hypervisor requires all Non-secure DMA capable devices that will be assigned to a Non-secure VM for direct control to be policed by stage 2 System MMU functionality. | Covered by Test 351. |
| 352 | B_SMMU_18 | If Secure-EL2 is not implemented, stage 2 System MMU functionality must be provided by a System MMU compatible with the Arm SMMUv2 specification or Arm SMMUv3 specification. | <p>1. Read out: ID_AA64PFR0_EL1.SEL2 .</p> <p>2. If enabled, and SMMU handles stage 2 policing, check whether SMMU_AIDR.ArchMajorRev \geq 3 and SMMU_AIDR.ArchMinorRev \geq 2.</p> <p>Else, check whether SMMU Major revision is at least 2.</p> |
| 353 | B_SMMU_19 | <p>When stage 2 System MMU functionality is provided by a System MMU compatible with the Arm SMMUv2 specification:</p> <ul style="list-style-type: none"> Each context bank must present a unique physical interrupt to the GIC. | <p>1. Create a list of all physical interrupts per context bank with their corresponding IDs.</p> <p>2. For each interrupt present in the list, check whether the assigned ID appears elsewhere in the list.</p> |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|--|
| - | B_SMMU_20 | <p>If Secure EL2 is implemented, stage 2 System MMU functionality that is made visible to a hypervisor must present the interface of a System MMU compatible with the Arm SMMUv3.2, or higher, architecture revision where:</p> <ul style="list-style-type: none"> The integration of the System MMUs is compliant with the rules in SMMUv3 integration, Section D [3]. SMMU implementations must provide level 1 or level 2 support for page table resizing. Arm recommends that the SMMU implements level 2. If the SMMU implementation provides level 2, then Arm recommends that the PE also provides level 2. | Covered by Test 352. |
| 354 | B_SMMU_21 | <p>When stage 2 System MMU functionality is provided by a System MMU compatible with the Arm SMMUv3 specification:</p> <ul style="list-style-type: none"> The integration of the System MMUs is compliant with the specification in Section D [3]. | Check whether SMMU_IDR0.COHAACC is set. |
| - | B_SMMU_22 | <p>MPAM architecture requires that all Requesters that can access an MPAM controlled resource, must support passing MPAM ID information.</p> <p>Therefore, a SMMUv3.2, or higher, implementation must support the MPAM extension if the requests the SMMU serves access MPAM-controlled resources.</p> | Covered by test 309. |
| 355 | B_SMMU_23 | If PE supports 16-bit VMID, the SMMU must implement support for 16-bit VMID. | <ol style="list-style-type: none"> Read out: ID_AA64MMFR1_EL1.VMID. If enabled, check whether SMMU_IDR0.VMID16 is set. |
| - | B_SMMU_24 | If Secure-EL2 is implemented, all secure DMA capable devices that can be assigned to a Secure VM must be policed by stage 2 Secure SMMU functionality. | <p>Not Implemented.</p> <p>It is not feasible to check if Secure-EL2 is implemented and get Secure devices configuration from ACS framework in UEFI shell.</p> |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|---|
| - | B_SMMU_25 | <p>If Secure-EL2 is implemented, stage 2 Secure MMU functionality must be provided by a system MMU compatible with the Arm SMMUv3.2, or higher, architecture revision where:</p> <ul style="list-style-type: none"> • The integration of the system MMUs is compliant with the rules in Section D [3]. • SMMU implementations must provide level 1 or level 2 support for page table resizing. | <p>Not Implemented.</p> <p>It is not feasible to check if Secure-EL2 is implemented from ACS framework in UEFI shell.</p> |
| - | SMMU_01 | All SMMU translation table walks and all SMMU accesses to SMMU memory structures and queues are I/O coherent (SMMU_IDR0.COHAAC == 1). | Covered by Test 354. |
| - | SMMU_02 | The system must be constructed so the act of the SMMU stalling on a fault from a device must not stall the progress of any other device or PE that is not under the control of the same operating system as the stalling device. | <p>Not Implemented.</p> <p>Require downstream device drivers to generate fault transactions that result in the stall of the SMMU. Not feasible to test this with the current ACS framework in UEFI.</p> |

2.6 Clock and Timer

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|---|
| 401 | B_TIME_01 | The base system must include the system counter of the generic Timer. | If the test can access the timer, the test is passed. |
| 401 | B_TIME_02 | The system counter of the generic timer must run at a minimum frequency of 10MHz. | <ol style="list-style-type: none"> 1. Read the CNTFREQ register. 2. If the read value is greater than 10MHz, test passed. 3. Else test failed |
| - | B_TIME_03 | The counter must not roll over inside a 10-year period. | <p>Not Implemented.</p> <p>RW access to CNTControlBase is allowed only through EL3. The BSA ACS is run on non-secure mode. Cannot change the register value from the tests.</p> |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|---|
| - | B_TIME_04 | The architecture of the counter mandates that the counter must be at least 56 bits, and at most 64 bits. From Armv8.4, for systems that implement counter scaling, the minimum becomes 64 bits. | Not Implemented. RW access to CNTControlBase is allowed only through EL3. The BSA ACS is run on non-secure mode. Cannot change the register value from the tests. |
| - | B_TIME_05 | This count must be available to the PE timers when they are active. When the PEs are in power state, the PE timer must be on. | Covered by wakeup and power tests. |
| 402 | B_TIME_06 | Unless all the local PE timers are always on, the base system must implement a system wakeup timer that can be used when PE timers are powered down. | 1. If any platform timer available, the test passes. 2. Else if all PE timers are always on, the test passes. 3. Otherwise, the test fails. |
| 403 | B_TIME_07 | The system wakeup timer must in the form of the memory-mapped timer. | 1. Check timer is secured or not. 2. For NS timer, check CNTLBASE is accessible or not 3. If CNTLBase is accessible, then timer is in the form of memory mapped timer. |
| 404 | B_TIME_08 | On timer expiry, the system wakeup timer must generate an interrupt that must be wired to the GIC as an SPI or LPI. Also, the system wakeup timer can be used to wake up PEs. | 1. Generate the timer interrupt. 2. Check if the interrupt is reaching GIC. 3. Check the system-specific interrupt controller with interrupt ID. |
| 405 | B_TIME_09 | The platform either implements hardware always-on PE timers or uses the platform firmware to save and restore the PE timers in a performance-scalable fashion. | 1. Program PE timer with a large value X . 2. Program system timer with a small value Y. 3. Put PE in to power down mode with PSCI call CPU_SUSPEND. 4. PE should wake up due to sys timer interrupt. 5. Read PE timer current count value, should be less than $((X - Y) + 1\% \text{ of } Y)$ & count should not be zero. |
| - | B_TIME_10 | If the system includes a system wakeup timer, this memory-mapped timer must be mapped on to Non-secure address space. | Covered by Test 403. |

2.7 Wakeup Semantics

Tests 501-505 are checking the BSA Table 8: Power state semantics.

Wakeup semantics rules RB_WAK_01 to RB_WAK_06 are derived from the above table.

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|--|---|
| 501-505 | B_WAK_01 | A PE must wake in response to a wakeup interrupt, independent of the state of its PSTATE interrupt mask bits, which are the A, I, and F bits, and of the wakeup interrupt priority. | <ol style="list-style-type: none"> 1. Suspend PE. 2. Get intid of wakeup interrupt. 3. Generate interrupt. 4. PE should wakeup irrespective of the mask bits. 5. Fail if does not wakeup. |
| - | B_WAK_02 | If the system supports a low-power state where the GIC is powered down, then there must be an IMPLEMENTATION DEFINED way to program the power controller to wake a PE on expiry of the system wakeup timer or the generic watchdog. In this scenario, the system wakeup timer or generic watchdog is still required to send its interrupt. | Not implemented in the current release. |
| 501-505 | B_WAK_03 | Whenever a PE is woken from sleep or off state the OS or hypervisor must be presented with an interrupt so that the PE software can determine the device that requested the wakeup. | <ol style="list-style-type: none"> 1. Install interrupt handlers. 2. Suspend PE. 3. Wake PE by providing interrupt. 4. Check if OS raises an interrupt once PE is woken. 5. Find which device has caused this interrupt. |
| 501-505 | B_WAK_04 | The interrupt must be pending in GIC at the point that control is handed back to the OS or hypervisor from the system-specific software performing the state restore. | <ol style="list-style-type: none"> 1. Generate interrupt. 2. Install interrupt handlers. 3. In ISR read the interrupt pending bit (it must be '1') 4. Clear interrupt and ISR. |
| 501-505 | B_WAK_05 | This interrupt must behave like any other, where a device sends an interrupt to the GIC, and the GIC sends the interrupt to the OS or hypervisor. The OS or hypervisor must not communicate with a system-specific interrupt controller. | <ol style="list-style-type: none"> 1. Generate the interrupt. 2. Check if the interrupt is reaching GIC. 3. Check the system-specific interrupt controller with interrupt id. |
| - | B_WAK_06 | If the wakeup event is an edge, then the system must ensure that this edge is not lost. The system must ensure that the edge wakes the system and is then delivered to the GIC without losing the edge. | Covered by Test 501-505. |

2.8 Power State Semantics

Tests 501-505 are checking the BSA Table 8: Power state semantics.

Power semantics rules RB_WAK_07 to RB_WAK_11 are derived from the above table.

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|---|
| - | B_WAK_07 | OS or hypervisor, or both, can be the reason for wakeup events and to know which timers are available to wake up the PE. All PEs must be in a state that is consistent with one of the semantics. | Covered by Test 501-505. |
| - | B_WAK_08 | System MMUs and GICv3 make use of tables in memory in the power states where GIC is on. For this type of state, system memory must be available and responds to requests without requiring intervention from software running on the PEs. | Covered in SMMU and GIC tests. |
| - | B_WAK_09 | When the system is in a state where the GIC is powered down, devices must not send messaged interrupts to the GIC. | Not implemented in the current release. |
| 501-505 | B_WAK_10 | Schematic check. | Covered by Test 501-505. |
| - | B_WAK_11 | Component check. | Covered by Test 501-505. |

2.9 Watchdog

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|---------|---|---|
| 701 | B_WD_01 | The generic watchdog must be implemented as specified in Section C of BSA specification 1.0 [3]. | Read and verify watchdog refresh and control registers. |
| - | B_WD_02 | The watchdog must have both its register frames mapped on to Non-secure address space. This watchdog is referred to as the Non-secure watchdog. | Covered by test 701. |
| 702 | B_WD_03 | Watchdog Signal 0 is routed as an SPI or an LPI to the GIC and it is expected that this is configured as a Non-secure EL2 interrupt, targeting a single PE. | <ol style="list-style-type: none"> 1. Generate the watchdog interrupt. 2. Check if the interrupt is reaching GIC. 3. Check the system-specific interrupt controller with interrupt ID. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|---------|--|---|
| - | B_WD_04 | Watchdog Signal 1 must be routed to the platform. | Not Implemented. WS1 signal is routed to a higher privilege entity to perform IMPDEF behavior. |
| - | B_WD_05 | The action taken on the raising of watchdog signal 1 is IMPLEMENTATION SPECIFIC. | Not Implemented. WS1 signal performs platform specific action. This is IMPDEF behavior. |

2.10 Peripherals

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|---|
| 601 | B_PER_01 | If the system has a USB2.0 host controller peripheral, it must conform to EHCI v1.1 or later. | 1. Get the BDF of USB. 2. Read the class code of the BDF from the config space and check if it confirms to the specification. 3. If not a PCIe device, then read the class code through firmware and check if it conforms to the specification. |
| 601 | B_PER_02 | If the system has a USB3.0 host controller peripheral, it must conform to XHCI v1.0 or later. | 1. Get the BDF of USB. 2. Read the class code of the BDF from the config space and check if it confirms to the specification. 3. If not a PCIe device, then read the class code through firmware and check if it confirms to the specification. |
| 602 | B_PER_03 | If the system has a SATA host controller peripheral, it must conform to AHCI v1.3 or later. | 1. Get the BDF of SATA device. 2. Read the class code of the BDF from the config space and check if it confirms to the specification. 3. If not a PCIe device, then read the class code through firmware and check if it confirms to the specification. |
| - | B_PER_04 | Peripheral subsystems which do not conform to rules B_PER_01, B_PER_02 or B_PER_03 are permitted if those peripherals are not required to boot and install an OS. | Manual verification required by user. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|---|
| 603 | B_PER_05 | <p>For system development and bring up, the base system must include a UART. The UART must be one of:</p> <ul style="list-style-type: none"> The Generic UART as specified in Section B [3]. <p>A fully 16550 compatible UART [9].</p> | <p>For generic UART algorithm steps are</p> <ol style="list-style-type: none"> 1. Install handlers to catch unexpected exceptions. 2. Get the base address for each UART present. 3. Validate the read-only registers UARTFR, UARTRIS, UARTRMIS, UARTRDR if they conform to BSA specification. <p>For 16550 UART, algorithm steps are:</p> <ol style="list-style-type: none"> 1. Get 16550 UART details from system hardware table (ACPI or DT). 2. Check if the I/O base address is present. 3. Check the baud rate from the hardware table. 4. Check the baud rate in the UART register. Obtain the divisor by enabling the divisor latch access and reading the divisor latch byte1 and byte2. Divisor = system clock speed / (16 * baud rate). 5. Check the read and write property of Line Control Register. 6 Check the read and write property of Interrupt Enable Register. 7 Check if UART is present using loopback test mode |
| 604, 606 | B_PER_06 | The UART interrupt output is connected to the GIC as an SPI or an LPI. | <ol style="list-style-type: none"> 1. Get the interrupt ID of the UART peripheral. 2. Install the ISR for the interrupt ID. 3. Generate an UART transaction. 4. Check if the interrupt is received in the ISR. |
| - | B_PER_07 | UART must be mapped on to Non-secure address space. This is called the Non-secure UART. | Covered by Test-604. |
| - | B_PER_08 | If the system has a PCI Express root complex, then it must comply with the rules in Section E. | Covered by PCIe tests. |
| 605 | B_PER_09 | <p>The memory attributes of DMA traffic must be one of the following:</p> <ul style="list-style-type: none"> Inner Write-Back, outer Write-Back, Inner Shareable. Inner non-cacheable, outer Non-cacheable. A device type. | <ol style="list-style-type: none"> 1. Get the DMA controllers present in the system. 2. Allocate a memory for a DMA transaction. 3. Read the memory attributes of the allocated memory. 4. If DMA supports coherent memory, then the attributes of the allocated memory should be inner/outer writeback, inner shareable. 5. If the DMA does not support coherent memory, then the attributes of the allocated memory should be inner/outer writeback inner shareable, inner/outer Non-cacheable transaction, or device type. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|---|
| - | B_PER_10 | I/O coherent DMA traffic must have the attribute - Inner Write-Back, Outer Write-Back, Inner Shareable. | Covered by Test 605. |
| - | B_PER_11 | If a TCG TPM-based security model is supported, the base system must provide a TPM implementation that is compliant to TPM library specification, Family 2.0 [4]. | Not implemented in the current release. |
| - | B_PER_12 | To ensure standard software support, a device claiming to follow the PCI Express specification [1] must follow all the rules in PCIe specification [1] which are software-visible | Covered by PCIe tests |

2.11 PCIe

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|---|
| 801 | PCI_IN_01 | Systems must map memory space to PCI Express Configuration Space, using the PCI Express Enhanced Configuration Access Mechanism (ECAM). | Read out the amount of detected ECAM regions. |
| 802 | PCI_IN_02 | Once the boot firmware hands control over to the operating system, application processor accesses to ECAM regions must work with no additional programming. The accesses must not require any OS visible programming. | For each ECAM region, access the PCI header space and extended PCIe configuration space. |
| - | PCI_IN_03 | The configuration space of all the devices, Root Ports, Root Complex Integrated Endpoints, and switches behind a PHB must be in a single ECAM region. | Part of device print info. Manual verification required by user by analyzing prints in the logs. |
| 803 | PCI_IN_04 | The configuration space of all the endpoints and switches in a Root port's hierarchy must be in the same ECAM space as the root port. | For all PCIe devices and switches in the system: 1. Find the Root Port they are under. 2. Check if the RP and the device are in the same ECAM region. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|---|
| - | PCI_IN_05 | <p>Root Port must appear as a PCI-PCI bridge to software (See Section 7.1 [1]). This implies that a Root Port:</p> <p>*Must have all registers that are part of the type 1 header, as specified in PCIe specification (See Section 7.5.1.3 [1]).</p> <p>* Must have all the capabilities required by PCIe specification for a Root Port. This includes the PCI express capability structure (See Section 7.5.3 [1]).</p> <p>* Registers must follow the access attributes (RW/RO, etc.) specified in the PCIe specification.</p> | <p>Register checks.</p> <p>Covered by Test 820-835.</p> |
| - | PCI_IN_06 | <p>PHB, in conjunction with Root Port, must recognize transactions that are coming in from application PEs as PCIe configuration transactions if the transaction address is within the ECAM range mapped to the Root Port, or the hierarchy that originates at that Root Port.</p> <p>This must be done by mapping the address of the incoming memory transaction to the PCIe Configuration address space, as described in Table 22 (See Section 7.2.2 [1]).</p> | Covered by PCIe table creation. |
| - | PCI_IN_07 | <p>PHB in conjunction with Root Port must return all 1s as read response data for Configuration read requests to nonexistent functions and devices on the root bus, that is the primary bus of the Root Port. No error must be reported to the software by the Root Port unless explicitly enabled to do so.</p> | Covered by PCIe Table creation. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|---|
| - | PCI_IN_08 | PHB, in conjunction with Root Port, must return all 1s as read response data for Configuration read requests that get an unsupported request response from downstream Endpoints or switches. No error must be reported to software by the Root Port unless explicitly enabled to do so. | Not implemented in current release. |
| - | PCI_IN_09 | PHB in conjunction with Root Port must return all 1s as read response data for configuration read requests that arrive at the Root Port when the Root port link is in DL_Down state (See Section 2.9.1 [1]). Note that this includes the case when the link is in L3 and the downstream device is in D3cold. No error must be reported to software by the Root Port unless explicitly enabled to do so. | Not Implemented. Controlling link state and device state through test sequence is not feasible without device specific drivers. |
| 909 | PCI_IN_10 | PHB in conjunction with Root Port must send out configuration transactions that are intended for the subordinate bus range of the Root Port as Type 1 Configuration transactions to downstream devices and switches. Subordinate bus range is between secondary bus number, exclusive, and the subordinate bus number, inclusive. | Increment the Root Port with exerciser endpoint connected subordinate bus by 1. Generate transaction by reading endpoint vendor id register. The endpoint must receive type 1 transactions. In case RP has a right sibling, set its secondary and subordinate bus number to invalid bus number so that transaction is not forwarded to it. Bus number restored at the end of test. |
| 910 | PCI_IN_11 | PHB in conjunction with Root Port must send out Configuration transactions that are intended for the secondary bus of the Root Port as Type 0 Configuration transactions to devices and switches downstream (See Section 3.2.2.3.1 [1]). | Initiate transaction for exerciser endpoint which is connected to the Root Port by reading its vendor ID. The exerciser must get transaction as type 0. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|---|
| - | PCI_IN_12 | PHB in conjunction with Root Port must recognize and consume configuration transactions intended for the Root Port configuration space and, read or write the appropriate Root Port Configuration register (See Section 3.2.2.3.1 [1]). | Covered as part of BDF table creation and PCIE tests. |
| 804 | PCI_IN_13 | PHB in conjunction with Root Port must recognize transactions received on the primary side of the Root Port PCI-PCI bridge, targeting prefetchable or non-prefetchable memory spaces of devices and switches that are on the secondary side of the bridge. | For all Root Ports in the system: 1. Detect the Non-Prefetchable address range. 2. Read / write memory from / to this memory range. |
| -- | PCI_IN_14 | PHB in conjunction with Root Port must return all 1s data to the requestor PE as the response for a configuration read if the following are true: <ul style="list-style-type: none"> • CRS software visibility is disabled or not present. • CRS response was received for the request the first time it was issued by the Root Complex. The Root Complex then tried to make the request return valid data by re-issuing the request an IMPLEMENTATION DEFINED number of times, but CRS was the response received for all such re-issues. | Not implemented. Generating CRS requests as part of test sequence is not possible on Silicon tests. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|---|
| -- | PCI_IN_15 | <p>PHB in conjunction with Root Port must return all 1s data to the requestor PE as the response for a configuration read if all the following are true:</p> <ul style="list-style-type: none"> • CRS software visibility is enabled. • The configuration read is not targeting the Vendor ID register. • CRS response was received for the request the first time it was issued by the Root Complex. The Root Complex then tried to make the request return valid data by re-issuing the request an IMPLEMENTATION DEFINED number of times, but CRS was the response received for all such re-issues. | <p>Not Implemented.</p> <p>Generating CRS requests as part of test sequence is not possible on Silicon tests.</p> |
| 808 | PCI_IN_16 | <p>PHB in conjunction with the Root Port must return all 1s data to the requestor PE as the response for a configuration read if the following are true:</p> <ul style="list-style-type: none"> • Target bus number of the request is not within the secondary bus to subordinate bus range of any of the Root Ports. • Target Bus, Device and Function (BDF) of the request does not match BDF of any on-chip functions. • Target BDF of the request does not match the BDF of any of the Root Ports. | <ol style="list-style-type: none"> 1. Get the maximum bus value from the PCIe info table. 2. Get highest BDF of the segment. 3. Get least high of max bus number. 4. Form BDF using the segment, bus, device, and function numbers. 5. Read should return all 1s. |
| 836 | PCI_IN_17 | The Root port must comply with the following (as per section 6.13 of [1]). | <p>Partially Covered.</p> <p>If ARI forwarding is disabled and target device number of the request > 0, then the access is terminated and all 1s data is returned to the requestor PE.</p> |
| 811 | PCI_IN_18 | The Root Port must comply with the byte enable rules that are specified in the PCIe specification (See Section 2.2.5 [1]) and must support 1 byte, 2 byte and 4-byte Configuration read and write requests. | <p>For every Root Port in the system:</p> <ol style="list-style-type: none"> 1. Read the command register of the RP with 8-bit (1 byte), 16-bit (2 byte), 32-bit (4 byte), and compare. 2. Verify the read and write behavior for each of 8-bit (1 byte), 16-bit (2 byte), and 32-bit (4 byte). |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|---|
| - | PCI_IN_19 | All registers present in the Root Port PCIe configuration space must follow the rules as defined in section 7.2 of the PCIe specification [1]. | Covered by other PCIe test. |
| 809 | PCI_IN_20 | Any vendor-specific data in the PCIe configuration space must be presented by one of the following capabilities, as defined in the PCIe specification [1]: <ul style="list-style-type: none"> • Vendor Specific Capability • Vendor Specific Extended Capability (VSEC) • Designated Vendor Specific Extended Capability (DVSEC). | For all the Root Ports in the system, search through the base and extended PCIe configuration spaces for non-PCIe compliant capabilities. |
| 861 | PCI_MM_01 | All systems must support mapping PCI Express memory space as device memory. | Map the BARs to normal memory attribute and check unaligned access and device memory attribute and check transaction |
| - | PCI_MM_02 | All systems must support mapping PCI Express memory space as non-cacheable memory. | Covered by Test 861. |
| - | PCI_MM_03 | When PCI Express memory space is mapped as Normal memory, the system must support unaligned accesses to that region. | Covered by Test 861. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|---|
| 865 | PCI_MM_04 | <p>Systems compliant to this specification must support 32-bit programming of NP BARs on such endpoints. This can be achieved in two ways:</p> <p>Method 1: PE physical address space can be reserved below 4GB, whilst maintaining a one-to-one mapping between PE physical address space and NP memory address space.</p> <p>Method 2: It is also possible to use a fixed offset translation scheme that creates a fixed offset in direction between PE physical address space, and PCI memory. This allows a window in PE physical address space that is above 4G to be mirrored in PCI memory space below 4G. This requires support in the PHB. Furthermore, firmware must program the PHB with the fixed offset, and to supply this information to the OS.</p> | <p>For the host bridge in the system, check the pre-fetchable type, if 0 then read the memory type.</p> <p>Scan all the bridge devices and check the memory type.</p> |
| 862 | PCI_MM_05 | For accesses from a PCIe endpoint to the host memory system, the address sent by PCI express devices must be presented to the memory system or SMMU unmodified. | <p>For all DMA requesters populated in the info table which are behind an SMMU, verify there are no additional translations before address is given to SMMU.</p> <p>Check if IOMMU ops is properly integrated for this device by making the standard OS DMA API call and verifying the DMA address is part of the IOVA translation table.</p> |
| - | PCI_MM_06 | For accesses from a PCIe endpoint to the host memory system, in a system where the PCI express does not use an SMMU, the PCI express devices have the same view of physical memory as the PEs. | Covered by Test 862. |
| - | PCI_MM_07 | For accesses from a PCIe endpoint to the host memory system, in a system with a SMMU for PCI Express there are no transformations to addresses being sent by PCI express devices before they are presented as an input address to the SMMU. | Covered by Test 862. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|--|
| 863 | PCI_MSI_1 | Support for Message Signaled Interrupts (MSI/MSI-X) is required for PCI Express devices. | Check if PCI device is PCI Express capable and MSI is supported |
| 904 | PCI_MSI_2 | The intended use model is that each unique MSI(-X) must trigger an interrupt with a unique ID and the MSI(-X) must target GIC registers requiring no hardware specific software to service the interrupt. | For all exercisers in the system, 1. Disable all SMMU and check if MSI-X capability is supported. If not supported, skip. Else, create MSI mapping and configure the MSI table. 2. Install ISR and trigger the interrupt. 3. Check completion of ISR. Clear the interrupt and the MSI mappings. |
| 865 | PCI_LI_01 | PCI Express legacy Interrupt messages must be converted to an SPI. | For all PCIe devices in the system: 1. Detect whether the legacy IRQ map exists. 2. If it exists, check whether the interrupt falls in SPI range. |
| 906 | PCI_LI_02 | A unique SPI ID must be allocated to each of the legacy interrupt lines of a PHB. It is permissible to share SPIIDs across PCI host bridges. | 1. Allocate memory for interrupt mappings. 2. Get the exerciser BDF. 3. Register an interrupt handler to verify legacy interrupt functionality. 4. Trigger the legacy interrupt. 5. Check the completion of interrupt service routine. 6. Return the interrupt. |
| 806 | PCI_LI_03 | Each legacy interrupt SPI must be programmed as level-sensitive in the appropriate GIC_ICFGR. | For all PCIe devices in the system: 1. Detect whether the legacy IRQ map exists. 2. If it exists, read GICD_ICFGR to check whether the interrupts in this map are level or edge sensitive. |
| -- | PCI_LI_04 | IMPLEMENTATION DEFINED registers must not be used to deliver these messages, only registers defined in the PCI Express specification and the Arm GIC specification. | OS boot covers this. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|--|
| -- | PCI_SM_01 | Hardware support for function or virtual function assignment to a VM or user space driver is optional, but if required must use a System MMU compliant with the Arm System MMU specification. | OS boot will cover this. |
| -- | PCI_SM_02 | Functions intended for VM assignment, or assignment to a user space driver must implement function level reset. | Covered by Test 835. |
| 907 | PCI_IC_01 | PCI Express transactions are not marked as No_snoop accessing memory that the PE translation tables attribute as cacheable and shared are I/O coherent with the PEs. | For all exercisers in the system, 1. Find the SMMU node and disable it globally so that the transaction passes through the SMMU without address modification. 2. Get a WB, outer shareable DDR Buffer. 3. Program exerciser hierarchy to start sending or receiving TLPs with No Snoop attribute header. 4. Initialize main memory region marked as WB, Outer Shareable by the PE page tables. CPU Write and read to this region with new data and ensure that the new data is cached in the CPU caches. Read and write the same data locations from the Exerciser with NS=0. The exerciser and CPU should get the latest data (hardware enforced cache coherency expected). |
| 864 | PCI_IC_02 | The PCI Express root complex is in the same Inner Shareable domain as the PEs. | The memory attribute of the PCIe Root complex is Inner Shareable. |
| - | PCI_IC_03 | This means that if a PCI Express device is accessing cached memory, then the transactions from the PCI Express devices will snoop the PE caches. | Covered by Test 907. |
| 908 | PCI_IC_04 | PCI Express also allows PCI Express devices to mark transactions as No_snoop. The memory accessed by such transactions must have coherency managed by software. | For all exercisers in the system: 1. Find the SMMU node and disable it globally so that the transaction passes through SMMU without address modification. 2. Get a WB, Outer Shareable DDR Buffer. 3. Program exerciser hierarchy to start sending or receiving TLPs with No Snoop attribute header. 4. Perform DMA transactions. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|--|
| 903 | PCI_IC_05 | If there is no SMMU or if an SMMU is not policing transactions from the Root Complex, the system must be able to distinguish between addresses that are targeted at memory and devices. Transactions that are targeted at devices must be treated as device type accesses. They must be ordered, must not merge, and must not allocate in caches. | <p>1. Read and Write on config space mapped to Device memory. Map config space to Arm device memory in MMU page tables. Perform transactions on incremental aligned address and on the same address.</p> <p>2. Read and write on BAR space mapped to Device memory. Map MMIO space to Arm device memory in MMU page tables. Perform transactions on incremental aligned address and on the same address.</p> |
| - | PCI_IC_06 | Transactions that are targeted at memory and that are marked No_snoop must be presented to the memory system as non-cached. Transactions that are targeted at memory and not marked as No_snoop must be presented as cached, shared. | Covered by Test 907. |
| - | PCI_IC_07 | If a memory page is marked as non-cached in the PE translation tables, all PCI Express transactions accessing that memory must be marked as No_snoop. Failure to do so can result in loss of coherency. | Covered by Test 907. |
| - | PCI_IC_08 | I/O Coherency Table. | Covered by Test 907. |
| - | PCI_IC_09 | In the case where the system has a System MMU translating and attributing the transactions from the Root Complex, the PCI Express transactions must keep the memory attributes assigned by the System MMU. If the System MMU-assigned attribute is cacheable then it is IMPLEMENTATION DEFINED if No_snoop transactions replace the attribute with non-cached. | <p>Not Implemented.</p> <p>This rule has IMPDEF behavior.</p> |
| - | PCI_IO_01 | If an implementation supports legacy I/O, it is supported using a one-to-one mapping between legacy I/O space and a window in the host physical address space. However, such schemes must not require a kernel driver to be set up, any necessary initialization must be performed before OS boot. | <p>Not Implemented.</p> <p>This rule requires an EP with legacy I/O support and device driver.</p> |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|---|
| - | PCI_IEP_1 | Anything claiming to follow the PCI Express specification must follow all the specification that is software-visible to ensure standard, quality software support. | iEP rules are out of scope for current release |
| | PCI_PP_01 | It is system-specific whether peer-to-peer traffic through the system is supported. | Not Implemented. This is IMPDEF rule. |
| - | PCI_PP_02 | Systems must not deadlock if PCI express devices attempt peer-to-peer transactions, even if the system does not support peer-to-peer traffic. This rule is needed to uphold the principle that a virtual machine and its assigned devices should not deadlock the system for other virtual machines or the hypervisor. | Not implemented in current release. |
| 819 | PCI_PP_03 | In a system where the PCIe hierarchy allows peer-to-peer transactions, the root ports in an Arm based SoC must implement PCIe access control service (ACS) features. | It is IMPLEMENTATION DEFINED whether a given platform supports peer-to-peer traffic. If the platform supports this, check in the ACS PCIe capability whether: <ul style="list-style-type: none"> 1. Source validation is supported. 2. Translation blocking is supported. 3. P2P request redirect is supported. 4. P2P completion redirect is supported. 5. Upstream forwarding is supported |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|---|
| 901, 902 | PCI_PP_04 | <p>For Root ports this means that the following must be supported:</p> <ol style="list-style-type: none"> 1. ACS Source Validation. (V) 2. ACS Translation Blocking. (B) 3. ACS P2P Request Redirect (R). 4. ACS P2P Completion Redirect (C). 5. ACS Upstream Forwarding (U). 6. The Root Port must support redirected request validation by querying an Arm architecture compliant SMMU to get the final target physical address and access permission information. 7. The Root Port must support ACS violation error detection, logging and reporting. Logging and reporting must be through the usage of AER mechanism. | <p>It is IMPLEMENTATION DEFINED whether a given platform supports peer-to-peer traffic. If the platform supports this, For an exerciser in the system.</p> <ol style="list-style-type: none"> 1. Get RP of the exerciser. 2. If ACS supported, Enable Source Validation & Transaction Blocking. 3. Find another exerciser on other root port, break from the test if no such exerciser is found. 4. If Both RPs Supports ACS, then check for ACS functionality. |
| 817, 818 | PCI_PP_05 | <p>If the Root port supports peer-to-peer traffic with other root ports, then it must support the following:</p> <ul style="list-style-type: none"> • Validation of the peer-to-peer transactions before sending it to the destination root port using the same mechanism as ACS redirected request validation. Any ACS violation error generated because of the request validation should be reported using the standard ACS violation error detection, logging and reporting mechanism specified in PCIe specification. • If the Root port supports Address Translation services and peer-to-peer traffic with other root ports, then it must support ACS direct translated P2P (T). | <p>It is IMPLEMENTATION DEFINED whether a given platform supports peer-to-peer traffic. If the platform supports this, then check:</p> <ol style="list-style-type: none"> 1. If ACS is supported, direct translated P2P is supported. 2. AER is supported |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|--|
| 905 | PCI_PAS_1 | If the system supports PCIe PASID, then at least 16 bits of PASID must be supported. This support must be full system support, from the root complex through to the SMMUv3 and any end points for which PASID support is required. | For all exercisers in the system behind an SMMU: 1. Create a mapping of 1 IOVA region to 2 PA regions, via SMMU (Each of the two mappings is identified by a distinct PASID). 2. Check whether there is support for at least 16-bit PASIDs. 3. Configure the exerciser DMA engine to access the IOVA region base. 4. Configure the exerciser to perform a DMA accesses with PASID1 in its transactions (Accesses must target PA region 1). 5. Configure the exerciser to perform DMA accesses with PASID2 in its transactions (Accesses must target PA region 2). |
| - | PCI_PTM_1 | Any system that implements PCIe Precision Time Measurement (PTM) [1] must use the Arm architecture defined System Counter [2] as PTM Requester time source at the PTM root(s). | Not Implemented in current release. |
| 814 | RE_PCI_1 | A RCiEP must obey all the rules that are specified in Section 1.3.2.3 of the PCIe 5.0 specification [1]. | Check if header type of an RCiEP is 0 and if it supports link capabilities by checking the link capabilities in the PCI Express capability structure. |
| 813 | RE_PCI_2 | The RCEC must obey all the rules that are specified in Section 1.3.4 of the PCIe 5.0 specification [1]. | Root Complex Event Collector has the Base Class 08h, Sub-Class 07h and Programming Interface 00h by reading the class code. Root Complex Event Collector Endpoint Association Extended Capability must be supported. |
| - | RE_CFG_1 | Ability to recognize read or write requests coming in from PEs as PCIe configuration requests if the requests address is within the ECAM range that is allocated to the host bridge of RCiEP. | Checked in multiple tests as part of read or write to BAR or config space. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|---|
| - | RE_CFG_2 | An IMPLEMENTATION DEFINED mechanism for providing the bus and device number to each RCiEP. | Manual verification required by user by checking number of RCiEP discovered. |
| - | RE_CFG_3 | An IMPLEMENTATION DEFINED mechanism for providing the bus and device number to each RCEC. | Manual verification required by user by checking number of RCEC discovered. |
| - | RE_CRS_1 | <p>When an RCiEP or RCEC function is temporarily unable to process a configuration request following a reset and the reset is one of the valid reset conditions defined in Section 2.3.1 of the PCIe specification [1], the following are response options:</p> <ul style="list-style-type: none"> Respond as defined in PCIe specification if the following are true: <ul style="list-style-type: none"> a) Configuration Retry Status (CRS) visibility is present and is enabled. b) The request is a configuration read to an address that includes the two bytes of the Vendor ID field. Otherwise, all 1s data must be sent back to the requestor PE. | <p>Not Implemented.</p> <p>On Silicon, we cannot place the end point into a CRS (not responding) state after reset on a need basis.</p> |
| 810 | RE_BAR_1 | All BAR registers in an RCiEP must be writeable and readable as per the PCIe specification. | <p>For every RCiEP:</p> <ol style="list-style-type: none"> Read the BAR value. Write a value to BAR register and read back. The original read and the read after write should not be same. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|--|
| - | RE_BAR_2 | Dynamic re-programming of these BAR registers must be allowed. | Covered by Test 861. |
| 834 | RE_BAR_3 | An RCiEP must not support I/O space claimed through BARs. | For every RCiEP device, 1. Read the BAR register. 2. Check if 32 bit or 64-bit BAR. 3. Check if BAR supports MMIO. |
| 816 | RE_INT_1 | RCiEP Endpoint must support MSI or MSI-X interrupt generation. | For every RCiEP device, check if MSI capability structure and MSI-X capability structure is supported. |
| 903 | RE_ORD_1 | The RCiEP must obey PCIe ordering rules for the configuration and BAR mapped memory spaces when accessed in the inbound direction, towards the RCiEP. | 1. Map the exerciser device BAR space to device memory. 2. Performs series of write or read transaction on incremental aligned address for 1, 2-, and 4-byte data. 3. Check the transaction are arrived in same order and not merged. Repeat same for configuration space of exerciser. |
| - | RE_ORD_2 | PCIe ordering rules must be obeyed while sending out completions for configuration space and BAR mapped memory space accesses. | Covered by Test 903. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|--|--|
| - | RE_ORD_3 | <p>If the RCiEP uses the PCIe producer-consumer model for the interaction with software or peer devices, then the following must be ensured by the RCiEP in collaboration with rest of the system:</p> <ul style="list-style-type: none"> • Write requests from the RCiEP are observed by other agents in the order required for the producer-consumer model to work. • A read request from the RCiEP must not overtake previously issued write requests from the same RCiEP if there is a read after write dependency between the read and previously issued write or writes. | <p>Not Implemented.</p> <p>On-silicon, unfeasible to verify the ordering of transactions</p> |
| - | RE_ORD_4 | The Transactions Pending bit must be cleared only after all outstanding reads, atomic requests, and write requests have received responses. | Not Implemented in current release. |
| | RE_SMU_1 | The RCiEP must interact with the SMMU in the same way that an external endpoint does. The rules to | Covered by PCIe tests. |
| 805 | RE_SMU_2 | PCIe ATS capability must be supported if the RCiEP has a software visible cache for address translations. | <p>Check if Address Translation Cache is present in this device.</p> <p>If ATC is present, check if ATS capability is present.</p> |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|---|
| - | RE_SMU_3 | PCIe PRI mechanism must be supported if RCiEP requires memory pages dynamically. | <p>Not Implemented in current release.</p> <p>Require RCiEP device driver to generate PRI requests and cannot determine if a RCiEP requires memory pages dynamically.</p> |
| 905 | RE_SMU_4 | If the RCiEP supports PASIDs, the PASID is used as SubStreamID as specified in the SMMU architecture specification. | <p>For each exerciser behind an SMMU, create a mapping of 1 IOVA region to 2 PA regions. Each of the 2 mappings is identified by a distinct PASID. Configure exerciser DMA engine to access IOVA region base.</p> <p>Configure exerciser to execute DMA with PASID_1 in transaction, access must be to PA region 1.</p> <p>Configure exerciser to execute DMA with PASID_2 in transaction, access must be to PA region 2.</p> |
| - | RE_SMU_5 | RCiEP must use its BDF to generate StreamID using rules that are described in Section H [3]. | <p>Not Implemented.</p> <p>Require RCiEP device driver to generate DMA transactions with the correct StreamID.</p> |
| 835 | RE_RST_1 | RCiEP must have Function Level Reset (FLR) support. | <ol style="list-style-type: none"> 1. Read FLR capability bit value. 2. If FLR capability is not set, move to next entry. 3. Allocate 4KB of space for saving function configuration space. 4. Get function configuration space address. 5. Save the function config space to restore after FLR. 6. Initiate FLR by setting the FLR bit. 7. Vendor Id should not be 0xFF after max FLR period. |
| 812 | RE_PWR_1 | RCiEP must have D state support and must have PCI Power management capability as specified in the PCIe specification. | For every RCiEP, check If Power Management capability is supported. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|----------|---|---|
| - | RE_PWR_2 | RCiEP must support PME messages for wake up signaling if the RCiEP must have a wake-up notification mechanism. | Not Implemented in current release. Require RCiEP device driver to generate PME messages. |
| - | RE_PWR_3 | PM_PME wake messages must be logged in the Root Complex Event collector that is associated with the RCiEP. | Not Implemented. Require RCiEP device driver to generate PME messages. |
| 815 | RE_ACS_1 | 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. | For every RCiEP device, check if following is supported: 1. Multifunction support. 2. P2P supported. 3. If ACS is supported, then AER should be supported. 4. Request Redirect P2P. 5. Completion Redirect P2P. 6. Direct Translated P2P. |
| - | RE_ACS_2 | If the RCiEP has ACS capability, then it must have AER capability for reporting ACS violation errors. | Covered by Test 815. |
| - | RE_ACS_3 | RCiEP requests that target peer endpoints must be mediated by the SMMU before proceeding to the target. | Not Implemented in the current release. Require RCiEP device driver to generate peer DMA transactions. |
| 820, 821 | RE_REG_1 | All type 0 header registers must be implemented for RCiEP and RCEC | Register checks |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|-----------------|
| 828, 829 | RE_REG_2 | All registers of the PCI power management capability must be implemented for RCiEP and RCEC. The registers must be implemented as described in Table 25 of BSA specification. | Register checks |
| 823-827 | RE_REG_3 | The registers that are specified in Section 7.5.3 of the PCIe specification [1] for all devices must be implemented for RCiEP. The registers must be implemented as described in Table 26. | Register checks |
| 820-833 | RE_REC_01 | The registers that are specified in Section 7.5.3 of the PCIe specification [1] for Root Complex Event Collectors must be implemented for RCEC. The registers must be implemented as described in Table 27. | Register checks |
| 820-833 | RE_REC_02 | All registers of the Root Complex Event Collector Endpoint Association Extended Capability must be implemented by the RCEC. The registers must be implemented as shown in Table 28. | Register checks |

2.12 DeviceID Generation and ITS Groups

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|---------|--|---|
| 276 | ITS_01 | An ITS group can contain one or more ITS blocks. | <ol style="list-style-type: none"> 1. Get number of ITS groups. 2. Check the number of ITS blocks in each ITS group. 3. If number of blocks < 1 in any group, the test fails. Otherwise, the test passes. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|---------|--|--|
| 277 | ITS_02 | An ITS block is associated with one ITS group. | <ol style="list-style-type: none"> 1. Get ITS id of first block in first group. 2. Check whether ITS id is repeating in other groups, if yes, the test fails. 3. Repeat this for all ITS blocks. |
| 911 | ITS_03 | A device that is expected to send an MSI is associated with one ITS group. | <ol style="list-style-type: none"> 1. Get ITS ID, device ID for current instance of exerciser. 2. Get Group Index of this ITS using get_its_info api. 3. Create ITS LPI mappings for this device, and fill MSI-X table. 4. Generate MSI and check if interrupt is raised. 5. Repeat 3-4 for all the ITS Blocks for this ITS Group. 6. Repeat 1-5 for all exerciser instances. |
| 911 | ITS_04 | Devices can be programmed to send MSIs to any ITS block within the group. | <ol style="list-style-type: none"> 1. Get ITS ID, device ID for current instance of exerciser. 2. Get Group Index of this ITS using get_its_info api. 3. Create ITS LPI mappings for this device, and fill MSI-X table. 4. Generate MSI and check if interrupt is raised. 5. Repeat 3-4 for all the ITS Blocks for this ITS Group. 6. Repeat 1-5 for all exerciser instances. |
| 912 | ITS_05 | If a device sends an MSI to an ITS block outside of its assigned group, the MSI write is illegal and does not trigger an interrupt that could appear to originate from a different device. See Section H.2.2 for permitted behavior of illegal MSI writes. | <ol style="list-style-type: none"> 1. Get ITS ID, device ID for current instance of exerciser. 2. Get Group Index of this ITS using get_its_info api. 3. Get ITS ID in an ITS group other than assigned ITS group. 4. Create ITS LPI mappings for this device, and fill MSI-X table. 5. Try to Generate MSI and check interrupt should not be raised. 6. Repeat 1-5 for all exerciser instances. |
| - | ITS_06 | An ITS group represents a DeviceID namespace independent of any other ITS group. | Not implemented in the current release. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|---|---|
| - | ITS_07 | All ITS blocks within an ITS group support a common DeviceID namespace size, a common input EventID namespace size and can receive an MSI from any device within the group. | Not implemented in the current release. |
| - | ITS_08 | All ITS blocks within an ITS group observe the same DeviceID for any given device in the same ITS group. | Not implemented in the current release. |
| - | ITS_DEV_1 | Every device requester that is expected to send MSIs has a DeviceID associated with it. | Not implemented in the current release. |
| - | ITS_DEV_2 | The system designer assigns a Requester unique StreamID to device traffic input to the SMMU. | Not implemented in the current release. |
| - | ITS_DEV_3 | When a device is not behind an SMMU, its DeviceID appears to high-level software as though it is assigned directly by the system designer. | Not implemented in the current release. |
| - | ITS_DEV_4 | The system must not allow this behavior to trigger an MSI that masquerades as originating from a different Requester. The system must anticipate that PEs also have the potential to be misused in this manner. | Not implemented in the current release. |
| - | ITS_DEV_5 | Every device that is expected to originate MSIs is associated with a DeviceID. | Not implemented in the current release. |
| - | ITS_DEV_6 | DeviceID arrangement and system design prevents any mechanism that any software that is not the most privileged in the system, for example VM, or application, can exploit to trigger interrupts associated with a different body of software, for example, a different VM, or OS driver. | Not implemented in the current release. |
| - | ITS_DEV_7 | If a device is a client of an SMMU, the associated DeviceID is derived from the SMMU's StreamID with an identity or simple offset function. | Not implemented in the current release. |

| Test Number | Rule ID | Scenario | Algorithm |
|-------------|-----------|--|---|
| - | ITS_DEV_8 | DeviceIDs derived from other kinds of system IDs are also created from an identity or simple offset function. For a Root Complex without an SMMU, the relationship is: DeviceID = zero_extend(RequesterID[N-1:0]) + (1<<N)*Constant_C | Not implemented in the current release. |
| - | ITS_DEV_9 | The relationships between a device, its StreamID and its DeviceID are considered static by OS or hypervisor software. If the mapping is not fixed by hardware, the relationship between a StreamID and a DeviceID must not change after system initialization, and OS drivers must not be required to set it up. | Not implemented in the current release. |

NOTE: PCIe iEP rules are out of scope for this release.

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] DEN0094A Arm® Base System Architecture version 1.0. Arm Ltd.

[4] PCI-To-PCI Bridge Architecture Specification 1.2. PCI-SIG.

Appendix A Revisions

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

Table A-1 Issue 01

| Change | Location |
|---------------|----------|
| First release | - |