RISC-V Server SoC Test Specification

Server SoC Task Group

Version v0.0.0, 2024-07-08: Draft

Table of Contents

Preamble

This document is in the Development state

Assume everything can change. This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.

Copyright and license information

Contributors

This RISC-V specification has been contributed to directly or indirectly by (in alphabetical order):

Aaron Durbin, Andrea Bolognani, Andrei Warkentin, Andrew Jones, Beeman Strong, Cameron McNairy, Greg Favor, Heinrich Schuchardt, Isaac Chute, Jon Masters, Ken Dockser, Krste Asanovic, Manu Gulati, Mark Hayter, Michael Klinglesmith, Paul Walmsley, Ravi Sahita, Shaolin Xie, Shubu Mukherjee, Sibaranjan Pattnayak, Ved Shanbhogue

Chapter 1. Introduction

The RISC-V Server SoC Test Specification defines a set of tests to verify if the requirements specified in RISC-V Server SoC specification are implemented. The tests specified in this specification are not intended to exhaustively verify the implementation. In most cases the tests only check for existence of the feature. Future versions of this specification may include more exhaustive tests.

The tests in this specification are documented use the following format:

TEST_ID#	Test algorithm
AB_CAT_NNN_MMM	The `CAT_NNN` identifies a requirement in the RISC-V Server SoC specification. Each requirement is associated with one or more tests identified by `MMM`. The test IDs are prefixed with two character prefix - `AB`. If character in position `A` is `M` then the test is for a requirement that MUST be supported and this test MUST pass. If character in position `A` is `O` then the test is for a requirement that SHOULD or MAY be supported; such tests may be skipped if the requirement is not implemented. The tests record if optional features were present in the test output log. The character in position `B` indicates the nature of the test. If this character is `F` then the test exercises some or all of the functionality associated with the feature. If the character is `E` then the test determines for evidence that the feature is implemented (e.g., check ACPI tables) but does not functionally exercise the feature.

This specification groups the tests in the following broad categories:

- Harts
- Clocks and Timers
- External Interrupt Controllers
- IOMMU
- PCIe subsystem
- Reliability, Availability, and Serviceability
- Quality of Service
- Debug
- Trace
- Performance monitoring
- Security

1.1. Glossary

Most terminology has the standard RISC-V meaning. This table captures other terms used in the document. Terms in the document prefixed by 'PCIe' have the meaning defined in the PCI Express

(PCIe) Base Specification [1] (even if they are not in this table).

Table 1. Terms and definitions

Term	Definition
ACPI	Advanced Configuration and Power Interface [2].
ACS	Follows PCI Express. Access Control Services. A set of capabilities used to provide controls over routing of PCIe transactions.
AER	Advanced Error Reporting. Follows PCI Express. A PCIe defined error reporting paradigm.
AIA	RISC-V Advanced Interrupt Architecture.
ATS	Follows PCI Express. Address Translation Services.
BAR or Base Address Register	Follows PCI Express. A register that is used by hardware to show the amount of system memory needed by a PCIe function and used by system software to set the base address of the allocated space.
BMC	Baseboard Management Controller. A motherboard resident management controller that provides functions for platform management.
CXL	Compute Express Link bus standard.
DMA	Direct Memory Access.
DMTF	Distributed Management Task Force. Industry association for promoting systems management and interoperability.
ECAM	Follows PCI Express. Enhanced Configuration Access Method. A mechanism to allow addressing of Configuration Registers for PCIe functions. In addition to the PCI Express Base Specification, see the detailed requirements in this document.
EP, EP=1	Follows PCI Express. Also called Data Poisoning. EP is an error flag that accompanies data in some PCIe transactions to indicate the data is known to contain an error. Defined in PCI Express Base Specification 6.0 section 2.7.2. Unless otherwise blocked, the poison associated with the data must continue to propagate in the SoC internal interconnect.
GPA	Guest Physical Address: An address in the virtualized physical memory space of a virtual machine.
Guest	Software in a virtual machine.
Hierarchy ID or Segment ID	Follows PCI Express. An identifier of a PCIe Hierarchy within which the Requester IDs are unique.
Host Bridge	Part of a SoC that connects host CPUs and memory to PCIe root ports, RCiEP, and non-PCIe devices integrated in the SoC. The host bridge is placed between the device(s) and the platform interconnect to process DMA transactions. IO Devices may perform DMA transactions using IO Virtual Addresses (VA, GVA or GPA). The host bridge invokes the associated IOMMU to translate the IOVA to Supervisor Physical Addresses (SPA).
HPM	Hardware Performance Monitor.

Term	Definition
Hypervisor	Software entity that controls virtualization.
ID	Identifier.
IMSIC	Incoming Message-signaled Interrupt Controller.
IO Bridge	See host bridge.
IOVA	I/O Virtual Address: Virtual address for DMA by devices.
MCTP	Follows DMTF Standard. Management Component Transport Protocol used for communication between components of a platform management system.
MSI	Message Signaled Interrupts.
NUMA	Non-uniform memory access.
OS	Operating System.
PASID	Follows PCI Express. Process Address Space Identifier: It identifies the address space of a process. The PASID value is provided in the PASID TLP prefix of the request.
PBMT	Page-Based Memory Types.
PRI	Page Request Interface. Follows PCI Express. A PCIe protocol that enables devices to request OS memory manager services to make pages resident.
RCiEP	Root Complex Integrated Endpoint. Follows PCI Express. An internal peripheral that enumerates and behaves as specified in the PCIe standard.
RCEC	Follows PCI Express. Root Complex Event Collector. A block for collecting errors and PME messages in a standard way from various internal peripherals.
RID or Requester ID	Follows PCI Express. An identifier that uniquely identifies the requester within a PCIe Hierarchy. Needs to be extended with a Hierarchy ID to ensure it is unique across the platform.
Root Complex, RC	Follows PCI Express. Part of the SoC that includes the Host Bridge, Root Port, and RCiEP.
Root Port, RP	Follows PCI Express. A PCIe port in a Root Complex used to map a Hierarchy Domain using a PCI-PCI bridge.
P2P or peer-to-peer	Follows PCI Express. Transfer of data directly from one device to another. If the devices are under different PCIe Root Ports or are internal to the SoC this may involve data movement across the SoC internal interconnect.
PLDM	Follows DMTF standard. Platform Level Data Model.
PMA	Physical Memory Attributes.
PMP	Physical Memory Protection.

Term	Definition
Prefetchable Non-prefetchable	Follows PCI Express. Defines the property of the memory space used by a device. For details see the PCIe Base Specification. Broadly, non-prefetchable space covers any locations where reads have side effects or where writes cannot be merged.
SMBIOS	System Management BIOS.
SoC	System on a chip, also referred as system-on-a-chip and system-on-chip.
SPA	Supervisor Physical Address: Physical address used to to access memory and memory-mapped resources.
SPDM	Follows DMTF Standard. Security Protocols and Data Models. A standard for authentication, attestation and key exchange to assist in providing infrastructure security enablement.
SR-IOV	Follows PCI Express. Single-Root I/O Virtualization.
TLP	Follows PCI Express. Transaction Layer Packet. Defined by Chapter 2 of the PCI Express Base Specification.
QoS	Quality of Service. Quality of Service (QoS) is defined as the minimal end-to-end performance that is guaranteed in advance by a service level agreement (SLA) to a workload.
UEFI	Unified Extensible Firmware Interface. [3]
UR, CA	Follows PCI Express. Error returns to an access made to a PCIe hierarchy.
VM	Virtual Machine.

Chapter 2. Server SoC Test Specification

2.1. Clocks and Timers

ID#	Algorithm
ME_CTI_010_010	Parse ACPI RHCT table to determine the time base frequency and verify it is equal to 1 GHz.
ME_CTI_020_010	Locate the _LPI objects for each RISC-V application processor hart. If present, verify that the *Architectural Context Lost Flags* have bit 0 set to 0, indicating no loss of timer context for each supported low-power idle state.

2.2. External Interrupt Controllers

ID#	Algorithm
ME_IIC_010_010	For each application processor hart: 1. Determine the ISA node in ACPI RHCT table for that hart. 2. Parse the ISA string in the ISA node and verify that Ssaia extension is supported. 3. Parse the RINTC structure in ACPI MADT tables to verify that the interrupt controller type for the hart is IMSIC.
ME_IIC_020_010	See ME_IIC_010_010.
MF_IIC_030_010	1. Verify presence of `siselect`, `sireg`, `stopi`, and `stopei` CSRs. 2. For each external interrupt identity supported by the S-level interrupt file, verify the ability to set the corresponding bit in the `eip`k and `eie`k registers. 3. Verify ability to enable and disable interrupt delivery in the `eidelivery` register. 4. Map the physical address of the S-mode interrupt register file of the hart with a virtual address using PBMT set to IO. The physical address is provided by the RINTC structure in ACPI MADT table. 5. Write a supported external interrupt identity to the S-level interrupt register file using a 4-byte store to the `seteipnum_le` register using virtual address established in previous step. 6. Read the `seteipnum_le` register using a 4-byte load to verify it reads 0. 7. Verify that the written external interrupt identity is recorded in the `eip`k register of the IMSIC. 8. Determine the highest priority pending and enabled interrupt in the `eip`k registers. 9. Read the `stopei` register to verify that the highest priority external interrupt identity is reported. 10. Clear any external interrupts pended or enabled in the IMSIC by this test by clearing the corresponding bits in the `eip`k and `eie`k registers.
ME_IIC_040_010	Use WARL discovery method on `hstatus.VGEIN` CSR field to determine the `GEILEN` and verify that at least 5 guest interrupt files are supported.
ME_IIC_050_010	Verify the number of supported supervisor mode interrupt identities in IMSIC structure of the ACPI MADT table is at least 255.
ME_IIC_060_010	Verify the number of supported guest mode interrupt identities in IMSIC structure of the ACPI MADT table is at least 63.
ME_IIC_070_010	See MF_IIC_030_010.

ID#	Algorithm
ME_IIC_080_010	1. Parse ACPI MADT to determine if an APLIC for supervisor interrupt domain is reported. 2. If no APLIC is reported then skip the remaining steps. 3. Locate the APLIC structure. 4. Verify that number of interrupt delivery control. structures is reported as 0 indicating it is used as a wired-to-MSI bridge. 5. Verify the `domaincfg` supports MSI delivery mode and is configured to be in MSI delivery mode. 6. Write an external interrupt ID to `genmsi` register and verify that the extempore MSI is delivered to the IMSIC of the targeted hart. 7. Verify that the guest index field of the `target[i]` registers support all values between 0 and `GEILEN` supported by the IMSIC.

2.3. Input-Output Memory Management Unit (IOMMU)

ID#	Algorithm
ME_IOM_010_010	1. Locate all IOMMUs reported by APCI and verify they are of RIMT type. 2. For each IOMMU, read the `capabilities` register and verify that it supports version 1.0 of the RISC-V IOMMU specification. 3. Output the `capabilities` register in the test output log.
ME_IOM_020_010	1. Use PCIe discovery to locate all RCiEPs and PCIe RPs. 2. Locate the ACPI RIMT tables of all IOMMUs 3. For each RCiEP, verify that there is a governing IOMMU. 4. For each RP, verify that there is a governing IOMMU.
ME_IOM_030_010	1. Locate all IOMMUs governing PCIe root ports. 2. For each located IOMMU: a. if `capabilities.MSI_FLAT` is 0, then the `ddtp` must support at least 2 level DDT. b. if `capabilities.MSI_FLAT` is 1, then the `ddtp` must support 3 level DDT.
ME_IOM_040_010	For each IOMMU that does not govern a PCIe root port: . Parse the ACPI RIMT structure of that IOMMU to determine the widest device ID. . Verify that the `ddtp` supports a mode that supports the widest device ID.
ME_IOM_050_010	1. Parse ISA string in ACPI RHCT table and determine the page based virtual memory systems supported by the harts. 2. For each IOMMU in reported: a. Verify that the `capabilities` register enumerates support for each of the page based virtual memory system modes supported by the harts.
OE_IOM_060_010	See ME_IOM_010_010.
OE_IOM_070_010	See ME_IOM_010_010.
ME_IOM_080_010	For each IOMMU, verify that if `capabilities.MSI_MRIF` is equal to `capabilities.AMO_MRIF`.
OE_IOM_090_010	See ME_IOM_010_010.
OE_IOM_100_010	See ME_IOM_010_010.
ME_IOM_110_010	1. Use PCIe discovery to locate all RCiEPs. 2. For each RCiEP: a. If PCIe ATS capability not supported by the RCiEP then continue. b. Locate the governing IOMMU using ACPI RIMT table. c. Verify that the `capabilities.ATS` is 1 in the governing IOMMU.
OE_IOM_120_010	See ME_IOM_010_010.
ME_IOM_130_010	For each IOMMU, verify that if `capabilities.IGS` is either 0 or 2.

ID#	Algorithm
ME_IOM_140_010	For each IOMMU, verify that if **fctl.BE** is either read-only zero or is writeable. Verify that the support is identical for all IOMMUs. If big-endian mode supported then emit the support status in the test output log.
OE_IOM_150_010	See ME_IOM_140_010.
OE_IOM_160_010	See ME_IOM_010_010.
ME_IOM_170_010	For each IOMMU, verify that if any of the **PD8**, **PD17**, or **PD20** bits are 1 in the **capabilities** register then **PD20** bit must be 1.
OE_IOM_180_010	See ME_IOM_010_010.
ME_IOM_190_010	For each IOMMU: 1. if **capabilities.HPM** is 0 then continue. 2. Verify **iohpmcycles** and its **OF** bit are writeable and the cycles counter is at least 40-bit wide. 3. Verify at least four programmable HPM counters are supported and the counters for each are at least 40-bit wide. 4. Verify that the bits corresponding to the implemented HPM counters in **iocountovf** and **iocountinh** are writeable. 5. Verify that the **iohpmcycles** is at least 40-bit wide. 6. Verify that the **CY** bit in **iocountovf** and **iocountinh** is writeable.
ME_IOM_200_010	See ME_IOM_090_010.
OE_IOM_210_010	See ME_IOM_010_010.
ME_IOM_220_010	1. Determine the width of the **PPN** field in **hgatp** and multiply that by 4096 to determine the PA size supported by the hart. 2. Verify that the **capabilities.PAS** is greater than equal to the PA size supported by the hart.
ME_IOM_230_010	No test.
OE_IOM_240_010	1. Do a PCIe scan to locate all RCiEP of IOMMU class and report the bus:device:function numbers of the IOMMUs in the test output log.
ME_IOM_250_010	No test.
ME_IOM_260_010	1. Parse the PCIe root complex device binding structures from ACPI RIMT table and build a mapping of root complexes associated with each IOMMU. 2. For each IOMMU determine the PCIe segment number of the associated PCIe root complexes and create a list of IOMMUs that govern multiple root complexes where the PCIe root complexes belong to two or more PCIe segments. 3. For each IOMMU that governs PCIe root complexes that are part of different PCIe segments verify that the **ddtp** supports 3 level DDT.
ME_IOM_270_010	No test.
OE_IOM_280_010	No test.

ID#	Algorithm
ME_IOM_290_010	No test.

2.4. PCIe Subsystem Integration

2.4.1. Enhanced Configuration Access Method (ECAM)

ID#	Algorithm
MF_ECM_010_010	1. Parse ACPI MCFG tables to local all ECAM ranges. 2. For each 4 KiB range in the ECAM range, verify that the following reads do not cause any errors or exceptions. a. 4-bytes at offset 0 - vendor and device ID b. 2-bytes at offset 0 - vendor ID c. 1 byte at offset 8 - revision ID
MF_ECM_020_010	1. Use PCIe discovery to locate the RISC-V PCIe test card. 2. Capture timestamp A from `time` CSR. 3. Write to `TEST_REG_1` in the test card DVSEC with a timeout value of 100 ns to request that the completion for CfgWr be generated after 1000 ns. 4. Issue a `fence iorw, iorw` instruction. 5. Capture timestamp B from `time` CSR. 6. Verify that the two timestamps are at least 1000 ns apart.
MF_ECM_030_010	1. Parse ACPI MCFG table and obtain ECAM ranges for all heirarchies. 2. Verify that the ECAM ranges for each hierarchy are all contigous and the base address is naturally aligned to the size. 3. Verify ranges of any two heirarchies do not overlap.
MF_ECM_040_010	See MF_ECM_030_010.
MF_ECM_050_010	TBA.
MF_ECM_060_010	1. This test requires an input parameter that indicates which primary bus number and root port can be used for this test. The test should be able to disable and enable the link associated with that root port without causing system instability (e.g., disabling link used to connect to boot device, etc.). Let the primary bus number be P and the RID of the root port be R. 2. Verify D is located on bus P. 3. Read vendor ID and device ID of all functions, including R, on bus P and record the results. 4. Disable the link using the link control register of R. 5. Read vendor ID and device ID of all functions on P and verify that they match values read before the link was disabled. 6. Enable the link using the link control register of R.
ME_ECM_080_010	For each PCIe root port in the system: 1. Read root capability register and verify that Configuration RRS Software Visibility is supported.

ID#	Algorithm
MF_ECM_090_010	1. This test takes the PCIe root port to which the test card is connected as an input parameter. 2. Increment the subordinate bus number of the root port. 3. Read the vendor ID of function on subordinate bus and verify that the PCIe test card receives a type 1 transaction. 4. Read the vendor ID of the test card on the secondary bus and verify that the PCIe test card receives a type 0 transaction. 5. Restore the subordinate bus number of the root port.
MF_ECM_100_010	1. This test requires an input parameters to use for the test: a. A primary bus number P. b. ECAM base address of the segment that includes P. c. The RID of a root port R on the primary bus P. d. The RID of a non-existent function NF on the bus P. e. The RID of a device D downstream of P that can be reset by the test. 2. Read PCIe header of R and verify it is of type 1. 3. Read vendor ID offset of NF and verify all 1's returned. 4. Write command register offset of NF and verify no errors or exceptions occur. 5. Make an unaligned 2 and 4 byte read to configuration space of R and verify all 1's returned. 6. Read PCIe header of D and verify it is of type 0 and note its vendor and device ID. 7. Disable CRS software visibility in R. 8. Issue FLR to D. 9. Read vendor ID of D and verify all 1's returned. 10. Keep reading vendor ID till D is discovered. 11. Enable CRS software visibility in R. 12. Issue FLR to D. 13. Read vendor ID of D and verify 0x0001 returned. 14. Read device ID of D and verify all 1s returned. 15. Keep reading vendor ID till D is discovered. 16. Disable link of R. 17. Read vendor ID of D and verify all 1's returned. 18. Enable link of R.
MF_ECM_110_010	See MF_ECM_100_010.
ME_ECM_120_010	No test.

2.4.2. PCIe Memory Space

ID#	Algorithm
ME_MMS_010_010	Use ACPI DSDT table to locate PCI host bridges and collect the memory ranges routed to each host bridge. Verify that each host bridge has a memory range available for use with 64-bit BARs and a memory range available for use with 32-bit BARs.
ME_MMS_020_010	See ME_MMS_010_010.
MF_MMS_030_010	1. Use PCIe discovery to locate the RISC-V PCIe test card. 2. Map the BAR 0 of the test card as IO memory using PBMT attribute. 3. Write data values to the `SCRATCHPAD` registers in BAR 0 using and read them back.
MF_MMS_040_010	This test requires the following inputs: a. A primary bus number P. b. ECAM base address of the segment that includes P. c. The RID of a root port R on the primary bus P. d. Changing the memory or prefetchable memory base/limit on R should not lead to any system instability i.e. R is not connected to the main NVMe/Network, etc. 1. Read the memory base/limit and prefetchable memory base/limit of the ranges bridged downstream of R. 2. Change limit to reduce the memory limit range by 1 MiB. Let this excluded 1 MiB range be E. 3. Perform 1, 2, 4, and 8 byte reads to locations in E and verify that all 1s is returned. 4. Perform 1, 2, 4, and 8 byte write to locations in E and verify that all no errors or exceptions occur. 5. Restore the memory limit and repeat same steps with the prefetchable memory limit. 6. Restore prefetchable memory limit to original value. 7. Disable link of R 8. Read 1, 2, 4, and 8 bytes from locations in memory base/limit range and prefetchable memory base/limit range and verify all 1s data returned. 9. Enable link R.
MF_MMS_050_010	See ME_MMS_040_010.

ID#	Algorithm
MF_MMS_060_010	This test requires the use of two functions of the RISC-V PCIe test cards. This test is optional if peer-to-peer DMA is not supported by the system. One test card function - card-f0 - is used as an initiator and the second card function - card-f1 - is used as a responder. 1. Disable poisoned TLP egress blocking in the root ports connecting to the test card. 2. Program the card-f0 to read from the `TEST_POISON_REG` in BAR 0 of card-f1. 3. Verify that card-f0 receives completion with EP=1.
MF_MMS_070_010	1. Use PCIe discovery to locate the RISC-V PCIe test card and memory map its BAR 0 as IO memory. 2. Read the MMIO register - `TEST_POISON_REG` in BAR 0 that responds with poisoned data. 3. Verify that either a hardware error exception occurs on the load instruction or the load returns all 1s data.
ME_MMS_080_010	1. For each PCIe root port, verify that no EA capability is reported.

2.4.3. Access Control Services (ACS)

ID#	Algorithm
ME_ACS_010_010	For each PCIe root port: 1. Verify ACS extended capability is supported. 2. Verify that the ACS capability register reports support for a. ACS source validation. b. ACS translation blocking. c. ACS I/O request blocking. 3. Report ACS capability register into test output log.
ME_ACS_020_010	For each PCIe root port: . If BAR0 or BAR1 are implemented, then verify that the ACS capability register supports ACS Enhanced Capability.
ME_ACS_030_010	No test.
ME_ACS_040_010	No test.
ME_ACS_050_010	No test.

2.4.4. Address Routed Transactions

ID#	Algorithm
MF_ADR_010_010	1. Use PCIe discovery to locate the RISC-V PCIe test card. 2. Configure IOMMU govering the test card with an invalid device context for the test card. 3. Initiate Translated Mrd, Untranslated Mrd, and a PCIe ATS translation request from the test card. 4. Verify that the IOMMU reports a "DDT entry not valid" fault for each of the transactions. 5. Verify that the test card receives a Unsupported Request (UR) response to each of the transactions.
MF_ADR_020_010	1. Use PCIe discovery to locate the RISC-V PCIe test card. 2. Determine the address range of system memory reserved for machine mode use. This may be a test input. 3. Configure the IOMMU govering the test card to allow read and write access to the reserved system memory range and enable use of ATS by the test card. 4. Initiate Translated Mrd and Untranslated Mrd request from the test card with an address in the reserved memory range. 5. Verify that the test card receives a Unsupported Request (UR) response to each of the transactions.
MF_ADR_030_010	This test requires the use of two functions of the RISC-V PCIe test card. One function - card-f0 - is used as an initiator and the second function - card-f1 - is used as a responder. 1. Program the IOMMU govering the card-f0 to remap the physical address range corresponding BAR 0 of card-f1 to an equally sized system memory buffer in page tables set up for card-f0. 2. Program card-f0 to initiate a memory read to an address in BAR 0 of card-f1 using an Untranslated request. 3. Verify that the data returned to card-f0 is from the corresponding offset in the system memory buffer and no transaction is received by card-f1.
MF_ADR_040_010	This test uses same setup as MF_ADR_030_010. 1. Program the IOMMU govering the card-f0 to disallow access to physical address range corresponding BAR 0 of card-f1 for DMA originating from card-f0. This test uses a Bare G-stage. 2. Program card-f0 to perform a memory read to the BAR 0 of card-f1. 3. Verify an Unsupported Request response is received and the IOMMU reports a "Read page fault". 4. Program card-f0 to perform a memory write to the BAR 0 of card-f1. 5. Verify that the IOMMU reports a "Write/AMO page fault".

ID#	Algorithm
MF_ADR_050_010	This test uses same setup as MF_ADR_040_010. 1. Program the IOMMU govering the card-f0 to allow read and write access to physical address range corresponding BAR 0 of card-f1 for DMA originating from card-f0. 2. Disable poisoned TLP egress blocking in the root ports connecting to the test card. 3. Program card-f0 to generate a Mwr to a test register in BAR 0 of card-f1 with EP=1. 4. Verify that card-f1 receives a Mwr with EP=1.
MF_ADR_060_010	1. Disable poisoned TLP egress blocking in the root ports connecting to the test card. 2. Program card-f0 to write a system memory location with poisoned data (EP=1). 3. Read the memory location written by the card-f0 from a RISC-V application processor hart and verify that a hardware error exception occurs. 4. Program card-f0 to read the previously written memory location and verify that the data is returned in a completion with with EP=1.
MF_ADR_070_010	See MF_ADR_060_010.

2.4.5. ID Routed Transactions

ID#	Algorithm
MF_IDR_010_010	1. Use PCIe discovery to locate the RISC-V PCIe test card. The test card must be connected directly to the root port. 2. Program the test card to generate a type 0 and a type 1 CfgRd request. 3. Verify that the card receives a Unsupported Request (UR) response.
OF_IDR_020_010	This test is optional and can be skipped if P2P routing of PCIe VDMs is not supported. This test requires the use of two functions of the RISC-V PCIe test card. One function - card-f0 - is used as an initiator and the second function - card-f1 - is used as a responder. 1. Program the card-f0 to generate a MCTP VDM with the card-f1 as the destination. 2. Verify that the VDM is received by card-f1.
OF_IDR_030_010	No tests.

2.4.6. Cacheability and Coherence

No tests are defined for these requirements.

2.4.7. Message signaled interrupts

A message signaled interrupt (MSI or MSI-X) is the preferred interrupt signaling mechanism in PCIe.

ID#	Algorithm
ME_MSI_010_010	1. Locate all RCiEP and PCIe root ports in the system and verify that the Interrupt Pin Register reads 0 indicating that the function does not use legacy interrupt messages. 2. Verify that all PCIe root ports support MSI and/or MSI-X capability.
ME_MSI_020_010	No test.
ME_MSI_030_010	See ME_MSI_010_010.

2.4.8. Precision Time Measurement (PTM)

ID#	Algorithm
OE_PTM_010_010	For each PCIe root ports, report the PCIe PTM capability if present in the test output log.
OE_PTM_020_010	No test.
OE_PTM_030_010	No test.

2.4.9. Error and Event Reporting

ID#	Algorithm
ME_AER_010_010	For each PCIe root port, verify that the AER extended capability is supported.
ME_AER_020_010	For each PCIe root port, verify that the DPC extended capability is supported.
ME_AER_030_010	For each PCIe root port, verify that the RP extensions for DPC is supported in the DPC extended capability.
OE_AER_040_010	For each RCiEP, report the presence of AER extended capability in the test output log.
ME_AER_050_010	For each RCiEP, determine if the ACS extended capability is supported and if supported verify that the AER extended capability is also supported.
ME_AER_060_010	If any RCiEP with AER extended capability were detected then verify that there is at least one RCEC in the root complex.
ME_AER_070_010	For each RCEC in the system: 1. Verify that it implements the RCEC endpoint association extended capability. 2. Verify that there is an RCEC associated with RCiEP with AER extended capability (See ME_AER_050_010).

2.4.10. Vendor Specific Registers

ID#	Algorithm
MF_VSR_010_010	1. Use PCIe discovery to locate all RCiEP, root ports, IOMMUs, and host bridges. 2. For each dicovered function walk the PCIe capability list and verify that the capability ID is one of PCIe specified capabilities.
MF_VSR_020_010	No tests.

2.4.11. SoC-Integrated PCIe Devices

ID#	Algorithm
MF_SID_010_010	No test.
MF_SID_020_010	For all RCiEP and root ports: 1. Verify that I/O BAR are not implemented. 2. Verify that no I/O EA capability are implemented. 3. Interrupt pin register reads 0.
MF_SID_030_010	No test.
ME_SID_040_010	For all RCiEP: 1. Verify that MSI-X extended capability is supported if SR-IOV extended capability is supported.
ME_SID_050_010	For all RCiEP: 1. Verify that if PASID extended capability is supported then the maximum PASID width supported is 20.
ME_SID_060_010	No test.
ME_SID_070_010	For all RCiEP: 1. Verify that if memory BAR are implemented then the BAR supports 64-bit memory space.
OE_SID_080_010	No test.
ME_SID_090_010	For all RCiEP: 1. Verify if ACS extended capability is supported then AER extended capability is also supported.
ME_SID_100_010	See ME_AER_050_010 and ME_AER_070_010.

2.5. Reliability, Availability, and Serviceability (RAS)

No tests are defined for these requirements.

2.6. Quality of Service

ID#	Algorithm
OE_QOS_010_010	1. Determine the ISA node in ACPI RHCT table for hart 0. 2. Parse the ISA string in the ISA node and report in test output log if Ssqosid extension is supported. 3. Determine if ACPI RQSC table is present and if present report support for CBQRI extension in test output log.
OE_QOS_020_010	See OE_QOS_010_010.
ME_QOS_030_010	If Ssqosid extension is supported, then verify that the `sqoscfg` CSR can hold at least 16 RCID and at least 32 MCID values.
OE_QOS_040_010	If CBQRI extension is supported, as determined by ACPI RQSC table, then report `capabilities.QOSID` bit of each IOMMU in the test output log.
OE_QOS_050_010	If ACPI RQSC table is not present then this test is skipped. 1. Determine caches in the Soc from the ACPI PPTT table. 2. Determine if there is a capacity controller implemented by that cache by looking up the cache ID in ACPI RQSC table and report in test output log whether capacity allocation and capacity monitoring are supported by that capacity controller by accessing the controllers capabilities register. 3. Locate all bandwidth controllers in ACPI RQSC table and and report in test output log whether bandwidth allocation and bandwidth monitoring are supported by that bandwidth controller by accessing the controllers capabilities register.
OE_QOS_060_010	See OE_QOS_050_010.
OE_QOS_070_010	See OE_QOS_050_010.
OE_QOS_080_010	See OE_QOS_050_010.
ME_QOS_090_010	If ACPI RQSC table is present then verify that the RCID and MCID count reported for all capacity and bandwidth controllers is identical in the quality of service controllers structures.
ME_QOS_100_010	No test.

2.7. Manageability

ID#	Algorithm
OE_MNG_010_010	1. Report into test log if a Management Controller Host Interface (Type 42) SMBIOS structures are present. 2. If Type 42 structures are present report into the test log the Device Type field indicating the type of network interface (USB, PCIe v2, etc.).
OE_MNG_020_010	1. Report into test log if a IPMI Device Information (Type 38) SMBIOS structure is present. 2. If Type 38 structure is present report into the test log the Interface Type (SSIF, etc.).
OE_MNG_030_010	No tests.

2.8. Performance Monitoring

These tests require the use of a vendor provided API to access the HPMs.

ID#	Algorithm
OF_SPM_010_010	1. Determine caches in the Soc from the ACPI PPTT table and obtain their cache IDs. 2. Allocate two regions of memory. 3. For each data cache: a. Use `CBO.FLUSH` to writeback and invalidate the two memory regions from the caches. b. Invoke vendor provided API, passing the cache ID as a parameter, to determine if the cache supports an HPM. c. If an HPM is supported then invoke the vendor provided API, passing the cache ID and events, to program the HPM. d. Perform a memory copy from one region to another from a hart that can access that cache. e. Use the vendor provided API to read the performance counters and verify that they update.
OF_SPM_020_010	1. Obtain the memory ranges from ACPI SRAT table and determine their proximity domains. 2. For each proximity domain: a. Allocate a region of memory in each proximity domain. 3. For each proximity domain - P: a. Use `CBO.FLUSH` to writeback and invalidate the memory regions from the caches. b. Invoke vendor provided API, passing the proximity domain as a parameter, to determine if the memory controller supports an HPM. c. If an HPM is supported then invoke the vendor provided API, passing the proximity domain and events, to program the HPM to count local/remote read/write bandwidth appropriately. d. Perform a memory copy from the region allocated in P to the region allocated in each of the other proximity domains from a hart with affinity to P. e. Use the vendor provided API to read the performance counters and verify that they update. 4. Repeat previous step but now copy to the region allocated in P from each of the other proximity domains and verify that the counters update.

ID#	Algorithm
OF_SPM_030_010	1. Use PCIe discovery to locate the RISC-V PCIe test card. 2. Determine the PCIe root port to which the card is connected. 3. Use vendor provided API, passing the test card and root port requester ID to determine if there is the PCIe port supports an HPM. 4. If an HPM is supported then invoke the vendor provided API, passing the PCIe root port RID and program the HPM to count read bandwidth. 5. Program test card to read from system memory. 6. Use the vendor provided API to read performance counters and verify they update. 7. Repeat previous steps but with counter programmed to count write bandwidth and the test card programmed to write to system memory.
OF_SPM_040_010	Use algorithm from OF_SPM_020_010.
OE_SPM_050_010	For all PCIe root ports, report in the test output log if the Flit performance measurement extended capability is supported.

2.9. Security Requirements

ID#	Algorithm
OE_SEC_010_010	For all PCIe root ports, report if the IDE extended capability is supported in the test output log.
OE_SEC_020_010	No tests.
OE_SEC_030_010	Report if system memory ranges are reported as crypto capable (EFI_MEMORY_CPU_CRYPTO) in the UEFI memory map.
OE_SEC_040_010	1. Report if the EFI TPM2 protocol is supported. 2. If EFI TPM2 protocol is supported, report the TPM present flag by retrieving the boot service capabilities.

Chapter 3. RISC-V PCIe Test Card

A PCIe test card is required to create stimulus for a subset of the tests. This section outlines the design of such a test card. The PCIe test card shall be designed to support a x1 PCIe lane and at least Gen 3 speeds. The test card shall be implemented as a single device with two functions. Each function implements the standard PCIe type 0 header and the following PCIe capabilities:

- PCIe Capability
- MSI Capability
- Power Management Capability
- ATS Capability
- DVSEC capability

Each function shall support a 64-bit memory BAR. The functions should support all legal values of Max_Payload_Size.

The DVSEC capability of the function shall provide the following registers:

- `TEST_REG_1` - a 32-bit wide test register that can be written with a value between 0 and 8191. When the register is written with a value of `X`, the function will generate a completion for the CfgWr transaction that writes the register after a delay of `X` nanoseconds.
- `TEST_REG_2` - a 32-bit wide register that may be configured with a value that represents the duration in nanoseconds that the device should wait before starting to respond to transactions following a FLR.

The BAR 0 of the functions provide the following registers:

- `TEST_POISON_REG` - a 64-bit wide test register which always returns a EP=1 completion on read.
- `SCRATCHPAD` - a 4 KiB read/write scratchpad register space that can be written using Mrd/Mwr transactions and supports all legal address and byte enable rules.
- `INJECTED_TLP` - a set of registers that can be used to setup the TLP header and if required the data payload of the TLP.
- `INJECT_TLP` - a control register to request the card to originate the TLP programmed in `INJECTED_TLP`.
- `LAST_RX_TLP` - a set of registers that hold the header and payload of the last TLP received by the function.

Bibliography

[1] “PCI Express® Base Specification Revision 6.0.” [Online]. Available: pcisig.com/pci-express-6.0-specification.

[2] “Advanced Configuration and Power Interface (ACPI) Specification.” [Online]. Available: uefi.org/specifications.

[3] “Unified Extensible Firmware Interface.” [Online]. Available: uefi.org/specifications.