SystemReady and SystemReady Pre-Silicon FAQ

Version 1.0

Non-Confidential

Issue 01

107981_0100_01_en

SystemReady and SystemReady Pre-Silicon FAQ

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	23 June 2023	Non-Confidential	Initial release

Proprietary Notice

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 Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

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

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. SystemReady and SystemReady pre-silicon Frequently Asked Questions

This guide provides answers to frequently asked questions about the Arm SystemReady program and SystemReady pre-silicon enablement.

The information is categorized into the following sections:

- SystemReady general FAQ answers general, high-level questions about the SystemReady program and SystemReady pre-silicon enablement. It includes information about the Base System Architecture (BSA) and Server Base System Architecture (SBSA) specifications.
- SystemReady compliance testing FAQ answers questions about how to test your design for compliance with the SystemReady requirements. It includes information about the tests, and how to run them.
- SystemReady exerciser FAQ answers questions about exercisers. Exerciser is the general Arm term for the component in the validation environment that provides external stimulus for comprehensive testing of your device. The exerciser is implemented as a controllable PCIe endpoints. It may be called a Transactor or VIP by EDA partners.

2. SystemReady general FAQ

This section provides answers to general questions about the Arm SystemReady program and SystemReady pre-silicon enablement.

2.1 What is SystemReady?

The Arm SystemReady certification program tests whether systems meet a set of hardware and firmware standards. Systems that meet the SystemReady standards give confidence that operating systems and subsequent layers of software “just work”. SystemReady certification therefore enables a rich and diverse set of software that runs seamlessly and securely across Arm-based devices.

You can think of SystemReady as a contract between silicon providers and Operating System (OS) developers. It defines which functionality the silicon provider must integrate into their design, and which functionality the OS developer must include in their software. SystemReady-compliant devices can therefore install and boot off-the-shelf operating systems without needing modification or patches.

SystemReady provides the following benefits for silicon partners:

- Reduced software costs. There is no need for kernel patches applied to Linux distributions, Board Support Packages (BSPs), or custom drivers.
- Faster time-to-market. A wide range of software is available to run immediately, without modifications.
- Wider addressable market. Silicon is no longer limited to a narrow software ecosystem.

Arm provides specifications that define the minimum required system functionality for a specific level of OS support. These specifications include the following:

- Arm Architecture Reference Manual (Arm ARM)
- Generic Interrupt Controller (GIC) specification
- System Memory Management Unit (SMMU) specification
- Base System Architecture (BSA) specification
- Base Boot Requirements (BBR) specification

The SystemReady certification program provides a suite of tests to check that the system under test has all required functionality. These tests report any deviations from the above specifications. This suite is called the Architecture Compliance Suite (ACS), available as an open-source project from GitHub.

2.2 What is Base System Architecture (BSA) compliance?

The BSA specification is an architecture specification for hardware compatibility. It defines the minimum CPU and system requirements to install and boot an off-the-shelf operating system. It specifies the hardware requirements for software-visible behavior that the OS, hypervisor, and firmware can rely on.

The BSA specification covers all the basic functionality that an OS or hypervisor requires. For more specific markets and applications, there are higher-level supplementary specifications which define the requirements of more specific systems, for example the Server Base System Architecture.

BSA compliance in the underlying platform hardware is needed for OS and hardware portability. This is central to the "software that just works" premise, allowing forwards and backwards compatibility between an OS and hardware.

Compliance with the BSA specification is key to the SystemReady certification program. It is a mandatory requirement for SR and ES certifications, and a recommendation for IR certification.

2.3 What is the difference between BSA and SBSA?

BSA is the Base System Architecture. It specifies the minimum requirements for off-the-shelf OS boot and install.

SBSA is the Server Base System Architecture, a supplement to BSA. It contains server-specific requirements for further software standardization. The SBSA specification collects features together into levels, from L3 to L7.

The BSA modules include the following:

- PE architecture features
- Memory map
- Interrupt controller
- PPI assignments
- SMMU
- Clock and timer subsystems
- Wakeup and power state semantics
- Watchdog
- Peripheral subsystem
- PCIe integration, RPs and EPs
- UART
- GIC and ITS

The SBSA, in addition to the above, has modules including the following:

- PCIe iEP and RCiEP
- RAS
- PMU
- Entropy
- MPAM

2.4 Which specification and level should I target?

The target compliance specification and level depend on the use case for the silicon and the target SystemReady certification band.

2.5 What is the SystemReady Pre-Silicon program?

The SystemReady Pre-Silicon program helps silicon vendors achieve compliance with BSA and SBSA before taping out. The SystemReady Pre-Silicon program helps provide a well-defined and low-risk path to SystemReady compliance.

The SystemReady Pre-Silicon program provides tools such as the pre-silicon BSA/SBSA compliance tests, and a framework with specific steps for silicon vendors to become BSA compliant.

BSA and SBSA are hardware specifications. This means that compliance must be achieved during the hardware design stage, pre-silicon. Integration and hardware BSA compliance issues are common, leading to software-visible defects and interoperability issues. Because of this, it is critical to design and test for pre-silicon compliance.

2.6 What are common compliance issues?

PCIe Enhanced Configuration Access Mechanism (ECAM), timers, and interrupts are challenging areas and often require hardware changes by silicon vendors and third-party IP vendors.

2.7 What is the impact of non-compliance?

Non-compliant silicon is not competitive. Mitigating hardware compliance issues in software often requires patches, custom operating systems, or firmware workarounds.

Developing these patches and workarounds is costly, time consuming, and not always feasible. Patches can be challenging to upstream and be approved by maintainers, hindering the vision of software that “just works”. The end-product may be suboptimal, uncompetitive, or not acceptable at all.

In the worst-case scenario, silicon that is not BSA-compliant may require an expensive and time-consuming re-spin, or risk non-adoption.

See the SystemReady Pre-Silicon Reference Guide for a comprehensive list of common compliance issues, and information about how to address and prevent them.

2.8 How can I design a BSA/SBSA compliant SoC?

During the different phases of the silicon design cycle, a silicon vendor designing for BSA/SBSA compliance must do the following:

Design phase	Task
Product definition and architecture exploration	Define the target SystemReady certification and BSA/SBSA compliance requirements for the design.
IP selection	Ensure that the silicon IPs used, whether designed internally or licensed from Arm or third parties, have the necessary features to enable BSA/SBSA compliance. This is particularly important for PCIe IP, therefore silicon vendors are strongly encouraged to request BSA/SBSA features from PCIe vendors.
Design and integration	Implement the rules as defined in the BSA/SBSA specifications.
Verification and bring-up	Test for pre-silicon BSA/SBSA compliance.

3. SystemReady compliance testing FAQ

This section provides answers to questions about compliance testing for the Arm SystemReady program and SystemReady pre-silicon enablement.

3.1 What is ACS? What are the compliance tests?

The Architecture Compliance Suite (ACS), also known as the compliance tests, include executable versions of the BSA and SBSA specifications. For more information, see BSA ACS and SBSA ACS. The tests check that the architectural rules are understood and implemented correctly.

More precisely, the compliance tests are a collection of self-checking, portable C-based tests. The tests are examples of the invariant behaviors that are defined by the BSA and SBSA specifications. Running the tests lets you confirm that these behaviors have been interpreted and implemented correctly in your design.

3.2 When can I run the compliance tests?

You can run the compliance tests during the pre-silicon phase and after tape-out.

In the pre-silicon phase, you run the compliance tests using RTL simulation or emulation. Running the pre-silicon tests provides confidence that the design is BSA/SBSA compliant before taping out.

After tape-out, passing the tests on silicon provides evidence that the end device is SystemReady compliant.

3.3 Is there a difference between the pre-silicon and on-silicon tests?

The compliance test suite and test cases are the same for both pre- and on-silicon testing.

During pre-silicon, silicon partners are strongly encouraged to run all the test cases, including the exerciser tests, to achieve complete BSA/SBSA compliance coverage.

For SystemReady certification, on-silicon BSA/SBSA compliance testing is a key requirement.

The pre-silicon compliance tests are a superset of the on-silicon tests, because the exerciser tests are not required for on-silicon. In the pre-silicon stage we can have these deeper tests, particularly of the PCIe integration, helping achieve complete compliance coverage.

Note

Arm only considers supporting firmware issues during SystemReady certification if the pre-silicon tests have been passed, including the exerciser tests.

3.4 How do I run the pre-silicon compliance tests?

The compliance tests are implemented in C, are self-checking and therefore portable across various platforms and environments.

During pre-silicon, they can be run, for example, on RTL simulation or emulation.

The tests can also be run on a model such as the Arm Fixed Virtual Platforms (FVPs). This is useful for early architecture exploration, building proof-of-concepts, and determining software integration requirements.

You can run the tests either on a UEFI shell or directly on bare metal.

3.5 Should I run the tests on UEFI or bare metal?

You can run the same compliance tests on either a UEFI shell or on bare metal, each of which bring different benefits:

- Running the tests on UEFI means that silicon designers can leverage existing firmware development work for faster porting and less integration effort. Arm provides pre-built ACS images that are portable across implementations and environments. EDAs may provide similar pre-built images that integrate their exerciser implementations.
- Running the tests on bare metal means that tests run earlier in the design cycle, before the firmware is ready. This results in shorter simulation and emulation cycles, and faster root cause analysis.

3.6 Which tests do I need to run? Are some tests optional or are all mandatory?

The Architecture Compliance Suite (ACS) has a built-in automated test execution mechanism. It runs all tests by default. However, if certain optional features or modules are absent, ACS skips those specific tests. Conversely, if mandatory features are missing, ACS accurately identifies them and marks corresponding tests as failures. This dynamic feature detection and test execution capability of ACS ensures seamless test execution based on the available system configuration.

3.7 How long does it take to run the tests?

The runtime of the PCIe component in an emulator is typically around 30 minutes, while running the entire suite takes approximately 10 hours.

The actual runtime depends on many factors, including system configurations such as the number of ECAM regions, PCIe devices, memory range, and the specific emulation or simulation run cycle.

3.8 How long does it take to port the ACS suite to my platform?

Porting the Arm Compliance Suite (ACS) to a bare metal system includes the following tasks:

- Integrating ACS into the bare metal boot flow
- Providing platform-specific information
- Implementing the necessary platform-specific PAL (Platform Abstraction Layer) APIs.

For detailed information about this porting process, see the Arm SBSA ACS Bare-metal User Guide.

In contrast, for UEFI-based systems, you only need to implement the exerciser PAL APIs. You can compile the ACS as a UEFI App, eliminating the need for extensive porting efforts. For more information about porting the exerciser PAL APIs, see the Exerciser Porting Guide.

3.9 How much does SystemReady certification cost?

At present, there are no direct costs associated with the SystemReady program. Arm does not charge a fee for SystemReady certification.

However, participation in the SystemReady certification program requires development effort to run tests and submit results, even if no issues need to be fixed.

EDA partners may charge for the tools and support they provide to assist with SystemReady compliance, for example exerciser implementations.

3.10 Are there any SystemReady success stories?

As of May 2023, Arm has issued over 100 SystemReady certifications for partner devices, and completed more than 10 SystemReady pre-silicon deployments. In addition, Arm products including the Morello demonstrator board and Corstone-1000 are now certified under the SystemReady program.

Partners who have achieved SystemReady certification include Marvell, NXP, Lenovo, Ampere, Raspberry Pi, and Arduino to name but a few.

For more information about SystemReady success stories, see the following resources:

- Arm SystemReady certifications reach 50 (May 2022)
- Progress in the Arm SystemReady Program (October 2021)
- The Arm SystemReady SR, ES, IR, and LS pages on the Arm website provide up-to-date lists of certified systems.

3.11 How do I bring up UEFI?

Porting UEFI to an Arm-based platform can be a complex and challenging task because of the large amount of source code involved. However, there are existing example ports available for popular Arm-based devices including BeagleBoard and Samsung Origen.

These example ports serve as ready-made references that can greatly simplify the porting process. For more information about these example ports, how they facilitate UEFI porting debugging, see UEFI Debug Made Easy.

3.12 What is the relationship between the BSA compliance tests and the PCI compliance tests provided by PCI-SIG?

The BSA tests cover the integration of PCIe in the system. The PCI-SIG tests cover the PCIe IP itself.

3.13 Do the compliance tests support multiple PCIe controllers?

The compliance tests provided by the Arm Compliance Suite (ACS) support multiple PCIe controllers.

The ACS suite scans the ECAM (Enhanced Configuration Access Mechanism) regions to detect and identify the PCIe controllers present in the system. Once identified, the PCIe tests are executed based on the specific configuration of each controller. This enables the ACS to effectively test the compliance and functionality of multiple PCIe controllers in a system.

3.14 How can I implement multiple ECAMs?

You can implement multiple ECAMs in either of the following ways:

- Separate hierarchies for each host bridge, with each having a bus number from 0 to 255
- All host bridges share a single ECAM region, with the bus numbers from 0 to 255 shared sequentially

3.15 Which Exception level should the tests run at?

The SBSA and BSA describe requirements on hardware and obligations on hypervisors and operating systems. Because of this, the ACS is expected to run at Non-secure EL2.

3.16 Does Arm support building UEFI on a partner platform?

Yes.

UEFI Debug Made Easy describes the process for porting and debugging UEFI.

For instructions to build and run UEFI on the AArch64 Foundation or Base FVPs (Fixed Virtual Platforms), see ArmPlatformPkg AArch64.

3.17 What are the reference designs available to run the ACS and where can I get them?

The Arm reference solutions are available from the following location:

- Arm reference solutions

For more information about how to run the ACS, refer to the SystemReady ES integration guide. In particular, the Set up the Raspberry Pi 4 section provides an example of how to build a SystemReady ES compliant platform.

3.18 Where can I get the latest build of the ACS?

The ACS is available as an open-source project from GitLab.

3.19 Should I run both the bare metal tests and the tests that are present in the Linux SBSA ACS?

The bare metal tests are a superset that include the UEFI-based tests, Linux-based tests, and also provide test coverage related to initializations such as PCIe, GIC and SMMU.

If you have run the bare metal tests, there is no need to run the Linux-based tests because they have already run.

3.20 What is the difference between pre-silicon, silicon, UEFI and bare metal tests?

UEFI is essentially a tiny operating system that runs on top of the boot firmware. It may be stored in flash memory on the motherboard, or it may be loaded from a hard drive or network share at boot.

UEFI tests can be executed on the UEFI Shell, functioning in both pre-silicon and silicon environments. Similarly, bare metal tests can also be performed in both pre-silicon and silicon environments, specifically in scenarios where no operating system is present.

4. SystemReady exerciser FAQ

This section provides answers to questions about exercisers, controllable PCIE endpoints that enable comprehensive testing of your device.

4.1 What is the exerciser? Why do I need one?

The exerciser PCIe end point is a crucial component of the validation environment used to develop BSA/SBSA PCIe exerciser tests.

It is a controllable PCIe endpoint device that can generate custom stimuli, legacy interrupts, MSI interrupts, and various DMA transactions. The exerciser is used by the BSA/SBSA ACS tests to provide external stimulus and generate different events that validate the PCIe Root Complex implementation on the test platform. This allows for deeper integration checks, and results in greater controllability and observability.

The exerciser is required to achieve complete BSA/SBSA compliance coverage, especially of the PCIe integration rules in BSA/SBSA.

This is because some of the BSA/SBSA PCIe integration rules require custom stimuli to test for compliance. This cannot be achieved solely using software running on the PE, and therefore an exerciser is required.

The exerciser tests compliance of PCIe integration and functionality, including the following:

- I/O coherency
- Snoop behavior
- Address translation
- PASID transactions
- DMA transactions
- MSI
- Peer-to-peer traffic
- Legacy interrupt behavior

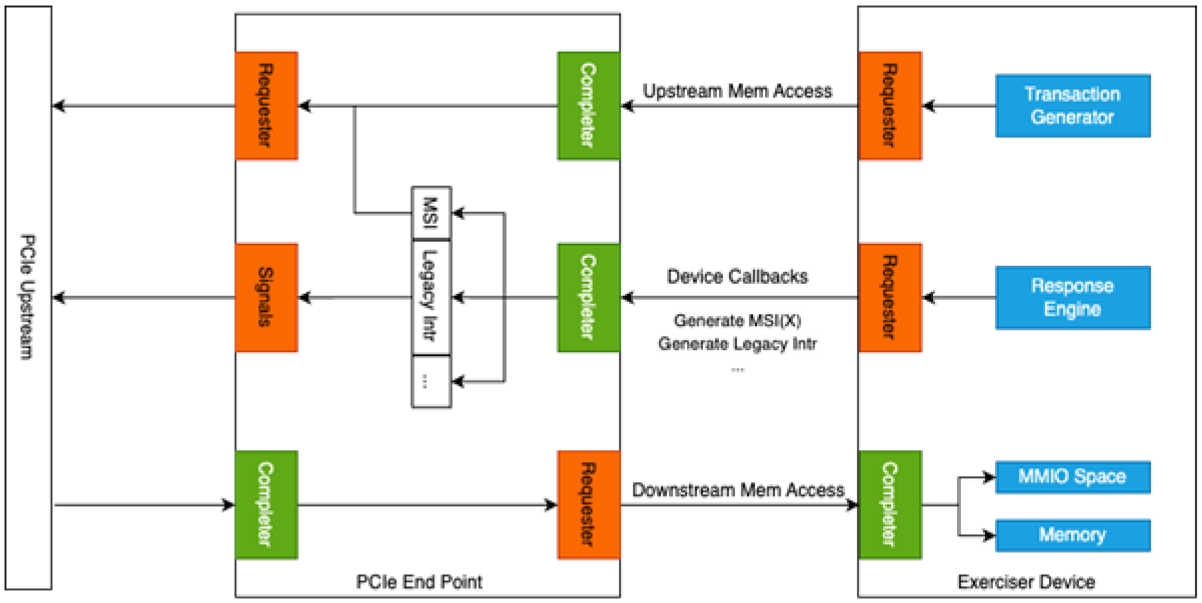
4.2 How does the exerciser integrate with the test environment?

Validating the compliance of specific PCIe rules defined in the BSA/SBSA specification requires the PCIe endpoint to generate specific stimuli during the runtime of the test. Examples of such stimuli are P2P, PASID, and ATC. The tests that require these stimuli are grouped together in the exerciser module.

The exerciser layer is an abstraction layer that enables the integration of hardware capable of generating such stimuli into the test framework. The exerciser PCIe endpoint is a PCIe endpoint device that can be programmed to generate custom stimuli for verifying the BSA/SBSA compliance of PCIe IP integration into an Arm SoC.

The following block diagram shows how the exerciser integrates with the test environment:

Figure 4-1: Block diagram for exerciser PCIe endpoint



4.3 How does the exerciser integrate with an Arm SoC?

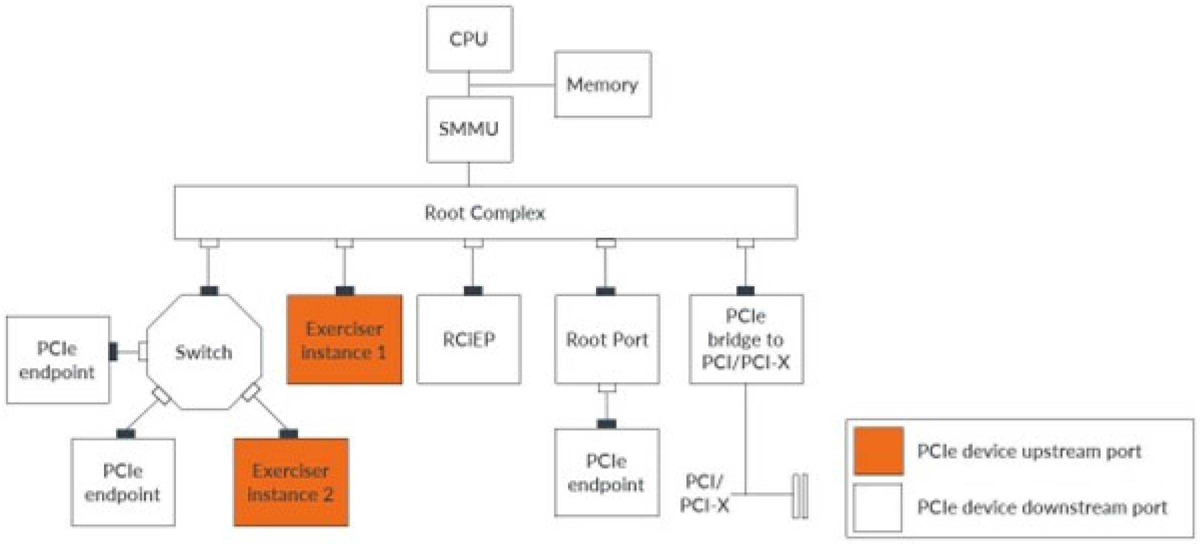
The following block diagram shows two instances of exerciser that are present in the system:

- Exerciser instance 1 is connected directly to the Root Complex as an RCiEP.
- Exerciser instance 2 is connected to the downstream port of a switch as a PCIe endpoint device.

Root Complex integrated Endpoint (RCiEP) and Root Complex Event Collector (RCEC) are endpoints connected directly to Root Complex.

PCIe endpoints are connected either to the Root Port or to downstream ports. Bridges connect PCI devices into the PCIe hierarchy while switches connect multiple PCIe devices to a single downstream port. PCIe devices access the GIC, memory, and the PE through the Root Complex, also called the Host Bridge.

Figure 4-2: Block diagram for exerciser representation in a system



4.4 How is the exerciser implemented?

The exerciser can be implemented, for example, as a PCIe Transactor or a Verification IP (VIP), that can be integrated into a pre-silicon simulation or emulation environment.

An exerciser must implement a specific set of required capabilities, as described in What capabilities must an exerciser support?.

4.5 Where can I get an exerciser? Does Arm provide one?

Arm defines the expected behavior of the exerciser in the Platform Adaptation Layers (PAL) API, which is part of the BSA/SBSA ACS.

EDA partners develop and commercialize exerciser implementations.

EDA partners may also provide more complete solutions for pre-silicon BSA/SBSA compliance. For example, these solutions might integrate the exerciser and the ACS with different verification, simulation, or emulation environments. EDA partners may also provide more comprehensive technical support.

Arm encourages silicon partners to contact their EDA representatives for information about how to license their exercisers and solutions. If necessary, Arm can introduce silicon partners to relevant EDA partner teams.

Note

EDA collaboration is the expected default, but silicon partners are free to source and use their own choice of tools if they wish.

4.6 Is the exerciser synthesizable?

The exerciser is a controllable PCIe endpoint that generates particular stimuli for test purposes.

EDA partners develop and commercialize exerciser implementations, but silicon partners are free to source and use their own choice of tools if they wish. If you obtain an exerciser from an EDA partner, the exerciser is available in that vendor's simulation environment. If you develop your own exerciser, you can implement it in whatever verification environment you choose.

Regardless of the source of the exerciser, if you have its RTL model then it should be technically synthesizable, but it is not needed in the final silicon.

4.7 What capabilities must an exerciser support?

The exerciser PCIe end point is a programmable device that generates custom stimuli for verifying the compliance of PCIe functionality in an Arm SoC.

EDA partners develop and commercialize exerciser implementations, but silicon partners are free to source and use their own choice of tools if they wish.

Regardless of the source of the exerciser, it must implement a specific set of required capabilities.

An exerciser implements the following capabilities:

- Generating legacy interrupts.
- Generating MSI interrupts.
- Generating a variety of DMA transactions.
- Generating specific stimuli such as P2P, PASID, and ATC for validating the compliance of certain PCIe rules defined in the BSA/SBSA specification.
- Emulating a bridge-bridge-endpoint hierarchy for testing enumeration sequence.
- Responding to configuration reads, specifically Vendor ID, without any prior enumeration.
- Responding with UR for non-existent functions.
- Working with and without ARI mode.
- Sinking type 1 and type 0 configuration requests.
- Responding with CRS response type for configuration accesses.

- The sinking configuration accesses an entire ECAM range and responds with either UR or normal Vendor ID based on user configuration.
- Reporting and checking the BDF and register address seen for each configuration address along with the access type, 1 or 0.
- Injecting an MSI interrupt with a specific target address, vector value, and requestor ID.
- Exposing multiple functions within an endpoint for testing enumeration.
- Injecting reads or writes to a specific address.
- Sending and receiving peer-to-peer transactions.
- Setting the AT bit to all its values.
- Sending error messages.
- Sending PME messages.
- Sending error responses.
- Checking the order in which incoming requests arrive.
- Checking the data that was written by an incoming write.
- Setting poison and receiving packets with the poison bit set.
- Injecting legacy interrupt messages.
- Sending memory access requests to the entire system address space, both 32-bit and 64-bit.
- Sinking inbound writes and reads.
- Sinking inbound configuration writes, reads, and responds correctly.
- Setting requestor IDs to specific values. The exerciser must be able to send transactions with different requestor IDs concurrently.
- Setting PASID to specific values. The exerciser must be able to send transactions with different PASIDs concurrently.
- Setting the No Snoop bit for a read or write request.
- Sending zero byte reads.
- Creating a link down condition so that the RP enters DPC.
- Performing DMA reads and writes.
- Resetting by a secondary bus reset or link disable.