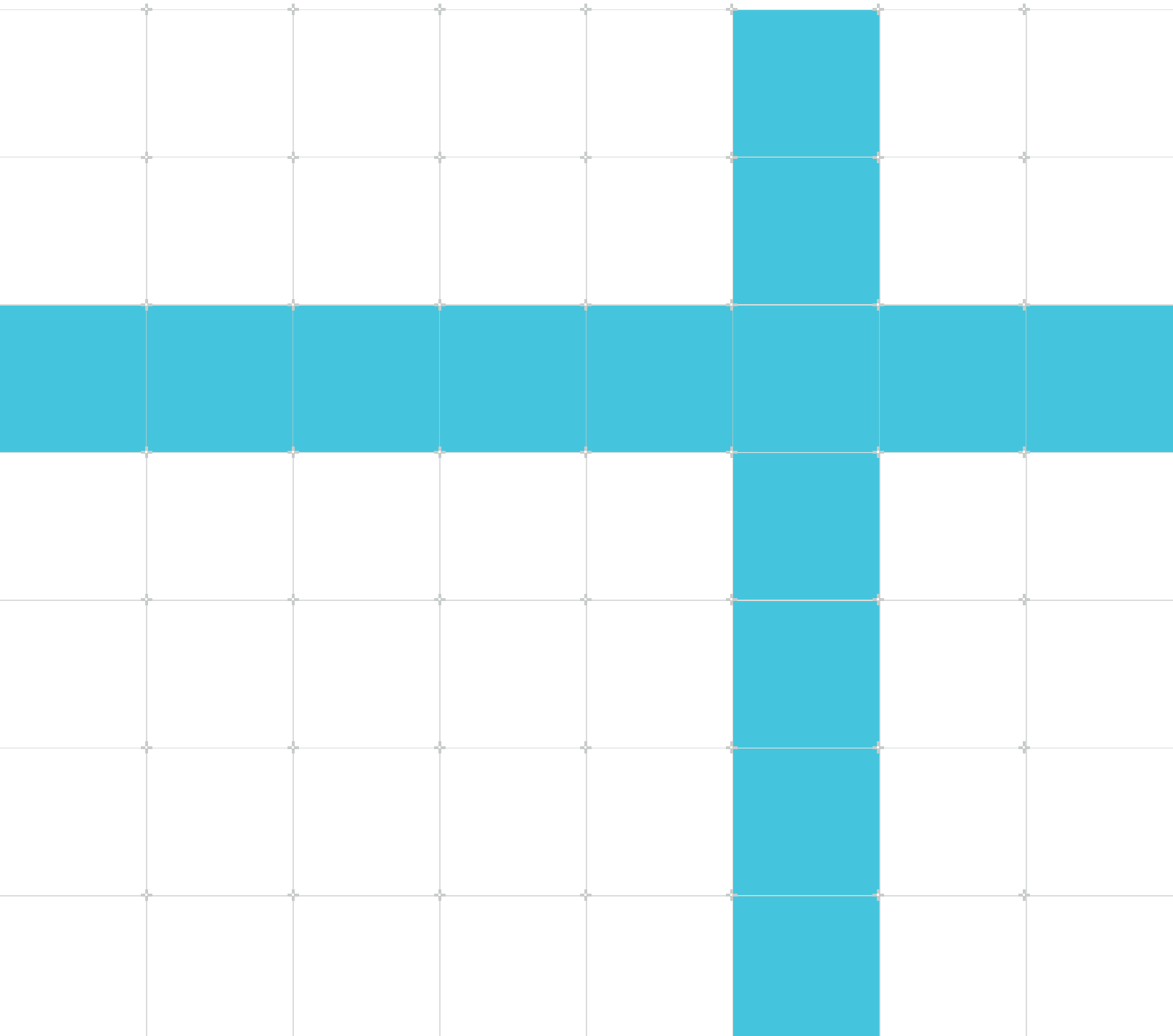
SystemReady FAQ

Version 3.5

Non-Confidential

Issue 01
107981_0305_01_en



SystemReady FAQ

Release information

Document history

Issue	Date	Confidentiality	Change
0305-01	6 February 2026	Non-Confidential	Major update, new SBMR chapter added
0304-01	21 October 2025	Non-Confidential	Major update, new chapters on SystemReady Devicetree band added
0303-02	16 July 2025	Non-Confidential	Minor update
0303-01	16 May 2025	Non-Confidential	Added “SystemReady band – General FAQ” and “Potentially waivable failures for SystemReady band compliance” chapters
0302-02	11 April 2025	Non-Confidential	Minor update
0302-01	11 April 2025	Non-Confidential	Updates to BBSR and Security Interface Testing FAQ
0301-01	31 January 2025	Non-Confidential	Minor update
0300-01	24 December 2024	Non-Confidential	Major update
0200-01	18 October 2024	Non-Confidential	Major update
0101-01	13 March 2024	Non-Confidential	Minor update
0100-01	23 June 2023	Non-Confidential	Initial release

Proprietary Notice

This document is protected by copyright and other related rights and the use 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 Limited ("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 the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. 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, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

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.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted 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 this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

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 Frequently Asked Questions

This guide provides answers to frequently asked questions about the Arm SystemReady program and compliance testing.

The information is categorized into the following sections:

SystemReady Program

The SystemReady Program - General FAQ answers general, high-level questions about the SystemReady program. It includes information about the Base System Architecture (BSA), Virtual Base System Architecture (VBSA), Server Base System Architecture (SBSA), and Base Boot Requirements (BBR) specifications.

SystemReady compliance testing

The SystemReady compliance testing – General 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.

The SystemReady band – General FAQ answers questions about SystemReady band self-declared compliance.

The SystemReady compliance testing - Pre-silicon FAQ answers questions about pre-silicon enablement and compliance testing to enable Base System Architecture (BSA) and Server Base System Architecture (SBSA) compliance.

The SystemReady compliance testing - 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.

The SystemReady compliance testing - BBSR and Security Interface Testing FAQ answers questions about BBSR and Security Interface Testing. It includes information on BBSR compliance tests and how to run them.

2. Glossary

This document uses the following terms and abbreviations.

Term	Meaning
ACPI	Advanced Configuration and Power Interface
ACS	Architecture Compliance Suites
BBR	Base Boot Requirements
BBSR	Base Boot Security Requirements
BMC	Baseboard Management Controller
BSA	Base System Architecture
CSP	Cloud Service Provider
CXL	Compute Express Link
DER	Distinguished Encoding Rules. A format that can be used for encoding keys.
EBBR	A recipe defined in the BBR specification requiring a subset of UEFI, also used for the Embedded Base Boot Requirements specification
FMP	Firmware Management Protocol
FWTS	Firmware Test Suite, see https://wiki.ubuntu.com/FirmwareTestSuite/
IBV	Independent BIOS or Firmware Vendor
IHV	Independent Hardware Vendor
ISV	Independent Software Vendor
LBBR	A recipe defined in the BBR specification supporting LinuxBoot.
OCP	Open Compute Project
ODM	Original Device Manufacturer
OEM	Original Equipment Manufacturer
OSV	Operating System Vendor
PCR	Platform Configuration Register
PK	Platform Key
PSA	Platform Security Architecture
QEMU	Quick Emulator
SBBR	A recipe defined in the BBR specification requiring UEFI and ACPI, also used for the previous Server Base Boot Requirements specification
SBMR	Server Base Manageability Requirements
SCT	UEFI Self-Certification Tests
SBSA	Server Base System Architecture specification. The BSA Supplement for Servers
SiP	Silicon Provider
SoC	System on Chip
SUT	System under test
TCG	Trusted Computing Group
TPM	Trusted Platform Module
UEFI	Unified Extensible Firmware Interface

For other terms and abbreviations, visit the Arm Glossary.

3. SystemReady Program - General FAQ

This section provides answers to general questions about the Arm SystemReady program.

3.1 What is SystemReady?

Arm SystemReady is a compliance program that helps ensure the interoperability of an operating system on Arm-based hardware. Developers can build software once and deploy it on any compliant Arm-based chip. The Arm SystemReady compliance program benefits the entire ecosystem because existing software can move through hardware generations and hardware providers can reach a broader addressable market.

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.

Arm provides architectural specifications for system hardware and firmware. These specifications include the following:

- Base System Architecture (BSA) specification
- Virtual Base System Architecture (VBSA) specification
- Server Base System Architecture (SBSA) specification
- Arm Base Boot Requirements (BBR) specification
- Server Base Manageability Requirements (SBMR) specification
- Base Boot Security Requirements (BBSR) specification

The SystemReady compliance program maintains a SystemReady Requirements Specification (SRS) which specifies a required level of adherence across these specifications according to context. This ensures that systems have a shared baseline across both hardware and firmware. The SystemReady program also provides suites of Architectural Compliance tests to check that a system under test has all expected functionality. Each suite identifies any deviations from the above specifications. These Architectural Compliance Suites (ACS) are collated into a single SystemReady live image for convenience, available as an open-source project on GitHub.

3.2 What is Base System Architecture (BSA) compliance?

The BSA specification is an architecture specification for hardware compatibility. The Base System Architecture (BSA) document specifies a minimum set of CPU and system architecture

requirements that are necessary for an OS to boot and run, regardless of the market segment. BSA includes baseline requirements for:

- Processor features
- Memory subsystem features
- GIC and SMMU revision features
- PCIe (Peripheral Component Interconnect Express) integration features
- Base levels for other peripherals, for example UART, USB, and SATA Security features
- Power Semantics

These requirements are intended to ensure that standard software, or operating systems, operate correctly on machines that are compliant with the BSA.

BSA compliance in the underlying platform hardware is needed for OS and hardware portability. Compliance with the BSA specification is key to the SystemReady band compliance, allowing forwards and backwards compatibility between an OS and hardware. It is a mandatory requirement for core SystemReady compliance band compliance, and recommended for SystemReady Devicetree compliance.

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.

3.3 What is the difference between BSA and VBSA?

Both BSA (Base System Architecture) and VBSA (Virtual Base System Architecture) define the minimum hardware requirements for Arm-based systems.

- BSA is intended for physical systems, such as servers, desktops, edge devices, and so on.
- VBSA is designed for virtual environments, such as virtual machines and cloud-based platforms.

3.4 What is the difference between BSA and SBSA?

BSA is the Base System Architecture. It specifies the minimum requirements for an interoperable Arm-based system.

The server supplement of the BSA is the Server Base System Architecture (SBSA). It contains server-specific requirements for further standardization. The SBSA specification collects features together into compliance levels, from L3 to L7. The different levels represent the advancement of the specification over time.

Generally, each level includes all the requirements of a previous level. For example, SBSA compliance Level 5 was added to correspond to new features in the Armv8.4-A architecture. Level

6 corresponds to changes in the Armv8.5-A architecture, and Level 7 corresponds to changes in the Armv8.6-A architecture.

The BSA covers 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

3.5 Which specification and level should I target for SystemReady compliance?

The SystemReady Requirements Specification (SRS) defines compliance bands with different expectations according to device enumeration approach. Each compliance band specifies the required and recommended adherence levels across a series of system architecture specifications. The target compliance specifications and levels therefore depend on the target use case and implementation strategy.

3.6 What is the BBR?

The Arm Base Boot Requirements (BBR) specification provides boot recipes that accommodate the different standards. The specification also provides the boot firmware implementations that

are used by a broader range of operating systems and hypervisors. Similar to the BSA, the BBR specification is extended with band-specific requirements.

The Base Boot Requirements (BBR) document specifies firmware interface requirements that system software, such as operating systems and hypervisors, can rely on. Firmware interface requirements include the following specifications:

- UEFI specification ACPI specification SMBIOS specification
- Other Arm specifications, for example, PSCI and SMCCC

The BBR also provides recipes, which are sets of requirements that are tailored to the various generic operating systems:

SBBR recipe:

- PSCI, SMCCC, UEFI, ACPI, SMBIOS
- SBBR compliance is a necessary requirement to achieve SystemReady band compliance

EBBR recipe:

- Requirements based on the EBBR specification
- PSCI, SMCCC, UEFI
- EBBR compliance is a necessary requirement to achieve SystemReady Devicetree band compliance.

For associated security requirements for a device to comply with industry standard security interfaces see What is BBSR?.

3.7 What are the SystemReady bands?

There are two SystemReady compliance bands targeting different user experiences:

- SystemReady band
- SystemReady Devicetree band

The SystemReady bands are versioned semantically with a major and minor version number.

The current versions and exact compliance requirements for the SystemReady bands are described in the SystemReady Requirements Specification. This should always be used to ensure information is current. Further information is also available at the Arm SystemReady Program Portal.

3.7.1 SystemReady band

The SystemReady band is the core compliance band for SystemReady. The previous SystemReady ES and SR bands are now merged into this SystemReady band. Also, The previous SystemReady VE can now be covered by SystemReady band.

The SystemReady band is designed for both physical systems and virtual environments that end users can install and run generic unmodified off-the-shelf standard operating system images, such as Windows, Linux, VMware, and BSD. It is market segment agnostic, providing a solution for servers, workstations, Data Processing Units (DPUs), Windows IoT devices, and other platforms or devices targeting generic off-the-shelf operating systems. By following a set of minimum hardware requirements and ACPI firmware abstraction forward and backward compatibility can be ensured, supporting old operating systems to run on new hardware and vice versa.

SystemReady band compliant systems must conform to:

- Base System Architecture (BSA) specification (If the system is a physical system)
- Virtual Base System Architecture (VBSA) (If the system is a virtual environments)
- SBBR recipe of the Base Boot Requirements (BBR) specification (UEFI, ACPI, SMBIOS, SMCCC, PSCI, and so on)
- Server Base System Architecture (SBSA) supplement specification (if the system is a physical server)
- Server Base Manageability Requirements (SBMR) (if the system is a physical server)

It is recommended that SystemReady band compliant systems should conform to:

- Base Boot Security Requirements (BBSR) specification

Note

servers must use Trusted Platform Module (TPM), and meet the TPM related requirements in the BBSR specification

---

For integration and testing guidance for the SystemReady band please refer to the Arm SystemReady Band Integration and Testing Guide.

For more information about how to perform self-declared SystemReady band compliance and the policy guidelines, please refer to the Arm SystemReady Band Policy Guidelines.

3.7.2 SystemReady Devicetree band

The Arm SystemReady Devicetree band meets the needs of the embedded Linux/BSD ecosystem on systems based on embedded Arm SoCs. It assumes a SoC supported by mainline Linux/BSD. It targets both custom images and pre-built images on IoT platforms. Some examples are

Yocto, OpenWRT, buildroot for custom and Debian, Fedora, SUSE for pre-built images. Forward compatibility, that is running old OS on new hardware, is not usually expected of these platforms.

SystemReady Devicetree band compliant systems must conform to:

- EBBR recipe of the Base Boot Requirements (BBR) specification
- Devicetree
- A subset of rules from the Base Boot Security Requirements (BBSR) specification

It is recommended that SystemReady Devicetree compliant systems should conform to:

- Base System Architecture (BSA) specification
- SMBIOS requirements of the Base Boot Requirements (BBR) specification
- Base Boot Security Requirements (BBSR) specification

For integration and testing guidance for the SystemReady Devicetree band please refer to the Arm SystemReady Devicetree Band Integration and Testing Guide.

For more information about how to perform self-declared SystemReady Devicetree band compliance and the policy guidelines, please refer to the Arm SystemReady Devicetree Band Policy Guidelines.

3.8 How does SystemReady collaborate with the industry?

The Arm ecosystem collaborates on the creation of standards with the Arm System Architecture Advisory Committee (SystemArch AC). This group includes companies across the Arm ecosystem, including OEMs, ODMs, silicon providers, OS vendors, software and firmware vendors, cloud service providers, and other hardware and IP vendors. The committee uses Causeway – an online collaboration platform for developing technical specifications – and communicates in various ways including email, regular conference calls, and biannual gatherings.

Any SystemArchAC member can raise a change request. Issues are described in the SystemArchAC regular meetings. When a change is agreed, it is integrated into the specifications.

Contact the SystemReady program via the Arm SystemReady Program Portal for further information.

4. SystemReady compliance testing – General FAQ

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

4.1 What is ACS? What are the compliance tests?

Each Architectural Compliance Suite (ACS) provides executable test coverage for an Arm system specification, such as the Base System Architecture (BSA) or Base Boot Requirements (BBR) specifications.

The compliance tests within each suite are examples of the invariant behaviors defined by each system specification. The compliance tests provided by the individual ACS enable you to check that the architectural rules stated in the specification are understood and implemented correctly

Each ACS is available in a dedicated repository, and the test suites are provided as collections of self-checking, portable C-based tests. Running the tests lets you confirm that these behaviors have been interpreted and implemented correctly in your design.

For example, compliance to the BSA and SBSA specifications, can be tested using BSA ACS and SBSA ACS in SYSARCH-ACS respectively.

4.2 When can I run the compliance tests?

You can run a compliance suite such as BSA ACS or SBSA ACS, during the pre-silicon design phase and the full set of compliance suites on a system-under-test after tape-out. Compliance tests associated to firmware specifications, such as BBR, can only be run effectively on a final system after tape-out.

In the pre-silicon phase, you run the hardware 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 full set of tests on silicon provides evidence that the end device is SystemReady compliant.

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

For SystemReady, depending on the compliance band being targeted, different specifications and considerations apply. While the Architectural Compliance Suites for BSA, SBSA, BBR, and SBMR, for example, can each be accessed and executed as standalone components, different combinations apply according to SystemReady compliance band expectations.

To give confidence that the expectations for the specific SystemReady compliance band have been met, the relevant test suites are collated and packaged into live OS images. This provides convenience for execution and evidencing across specifications. Each image is tailored to support testing and evidencing for the specific requirements of the associated SystemReady band.

Each SystemReady live image has a built-in automated test execution mechanism. All tests are run by default. However, if certain optional features or modules are absent, those specific tests are skipped. Conversely, if mandatory features are missing, the corresponding tests are marked as failures.

Each SystemReady Live OS image is comprised of a set of UEFI applications on UEFI shell and a Linux kernel with a simple BusyBox based file system. The Firmware Test Suite (FWTS) is integrated into the file system. The FWTS is a package hosted by Canonical which provides tests for ACPI, SMBIOS, Device tree and UEFI.

Each image is a “live” image, which means it does not need installing onto the system under test. The image, when flashed onto boot media like a USB drive, SD-Card, or NVMe drive can then be booted on the System Under Test (SUT).

These SystemReady prebuilt live OS images can be downloaded from the dedicated Arm-SystemReady repository on GitHub.

4.4 How long does it take to run the tests?

The actual runtime depends on many factors, including:

- The number of tests
- The test suites being executed
- Your testing environment
- The system configuration

For example, changes to system configurations such as the number of ECAM regions, PCIe devices, memory range, and the specific emulation or simulation run cycle all impact the time taken to complete a testing cycle.

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

The runtime of the BBR test (SCT) on a USB drive or SD-Card is typically longer than 6 hours. Therefore Arm recommends that you flash the live image to a NVMe/SATA drive or USB disk enclosure with a fast SSD drive.

4.5 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 the following:

- RD-N2 FVP
- Raspberry Pi 4

These example ports serve as ready-made references that can greatly simplify the porting process.

For more examples and information about porting, see EDK II Sample Platforms.

For more information about debugging, see EDK II Debugging.

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

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

4.7 Do the compliance tests support multiple PCIe controllers?

The compliance tests provided by the Architecture Compliance Suites (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 each ACS to effectively test the compliance and functionality of multiple PCIe controllers in a system. This is only required for the SystemReady band.

4.8 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

4.9 Which Exception level should the BSA/SBSA tests run at?

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

4.10 Does Arm support building UEFI on a partner platform?

Yes.

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

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

Each System Architecture Compliance Suite (ACS) is available in a dedicated repository, where the latest releases can be accessed.

The SystemReady prebuilt live OS images contain the latest builds of all relevant ACS. These can be downloaded from the Arm-SystemReady repository on GitHub.

4.12 How do I setup to run the SystemReady live image?

Pre-built SystemReady live ACS images for each SystemReady compliance band are available on Github:

- For the SystemReady band, https://github.com/ARM-software/arm-systemready/tree/main/SystemReady-band/prebuilt_images
- For the SystemReady Devicetree band, https://github.com/ARM-software/arm-systemready/tree/main/SystemReady-devicetree-band/prebuilt_images

This live image is deployed onto a storage device, such as a USB drive, which can then be used on the System under Test.

You must do the following before running the SystemReady ACS Live Image:

- Make the bootable ACS live image available on the System under Test (SuT) on a bootable storage device.
- Prepare the SuT machine with up-to-date firmware and a host machine for SuT console access.
- For checking BBSR compliance (UEFI secure boot), make sure the system firmware is in Setup Mode. This is where the Secure Boot keys are cleared before starting the ACS. The mechanism to enroll Secure Boot keys is platform-specific and the procedure to enroll the keys must be available.
- If the system only supports in-band system firmware updates, you must run the capsule update ACS test. A vendor-provided firmware update capsule must be available on a storage device on the system.

The documentation for the band being tested provides further guidance for deployment:

- For the SystemReady band follow the deployment steps in the SystemReady Band Integration and Testing Guide
- For the SystemReady Devicetree band follow the deployment steps in the SystemReady Devicetree Band Integration and Testing Guide

Further guidance according to compliance band is also available on GitHub: https://github.com/ARM-software/arm-systemready.

A typical deployment process consists of the following steps:

- Uncompress the prebuilt ACS image.
- On the Linux host machine, deploy the prebuilt ACS image to the storage device using one of the following sets of commands. The image name and path should reflect the image being deployed.

  For SystemReady band:

  ```
  $ lsblk
  $ sudo dd if=/path/to/systemready_acs_live_image.img of=/dev/sdX
  $ sync
  ```

  For SystemReady Devicetree band:

  ```
  $ lsblk
  $ sudo dd if=/path/to/systemready-dt_acs_live_image.wic of=/dev/sdX
  $ sync
  ```

  The `lsblk` command displays the storage devices in the system. Use the output of this command to identify the name of the target storage device.

  The `dd` command copies the prebuilt ACS image to the storage device. Replace `/dev/sdX` with the name of the target storage device.

The `sync` command forces the copy operation to complete, so that you can unplug the target storage device.

4.13 What does the SystemReady ACS live image testing process look like?

For information about the test process using the SystemReady ACS Live Image, including process flowcharts, refer to the corresponding Integration and Testing Guide:

- For Arm SystemReady band, see the "ACS" chapter in the Arm SystemReady Band Integration and Testing Guide.
- For Arm SystemReady Devicetree band, see the "Test with the ACS" chapter in the Arm SystemReady Devicetree Band Integration and Testing Guide.

4.14 How do I get support with SystemReady compliance?

Arm provides the following:

- This guide, which answers common issues across the SystemReady program.
- The Arm SystemReady Program Portal which provides links to compliance bands, guides, videos and more information.
- The SystemReady Support Forum where queries can be raised.
- The Arm SystemArchAC mailing list. Contact the SystemReady program via the Portal for further information.
- ACS support. Send an email to support-systemready-acs@arm.com.

5. SystemReady band – General FAQ

This section provides answers to general questions about the Arm SystemReady Band compliance.

5.1 Why has Arm transitioned SystemReady SR and ES certifications to SystemReady band self-declared compliance?

Arm has transitioned SystemReady SR and ES certifications to SystemReady band self-declared compliance because the SystemReady SR and ES certifications have successfully fulfilled the SystemReady program's initial objectives.

Now it is time to move to a more sustainable approach, which is the SystemReady band self-declared compliance.

The reasons for this move include the following:

1. Maturity of the ecosystem and market:
    - Key industry players including major Silicon Providers (SiPs), Cloud Service Providers (CSPs), Independent BIOS Vendors (IBVs), Original Design Manufacturers (ODMs), and Original Equipment Manufacturers (OEMs) are actively involved, resulting in a significant increase in SystemReady SR certified and potentially compliant systems over the past two years.
    - Certifications from Operating System Vendors (OSV), for example Redhat, SUSE, and Ubuntu, and a Windows 11 image for Arm-based systems have become publicly available over the past few years
2. Challenges of a mature ecosystem:
    - With the increase in requests, the certification process itself has become a bottleneck that slows down partners' marketing activities.
    - Duplication of efforts across different SKUs (Stock Keeping Unit) and new releases that are based on the same SoC.
    - Concerns from partners about the resources needed to support operating systems required solely for certification purposes, when those operating systems are not intended for actual use or officially supported by the partner.
    - Confusion between SystemReady certification and OS-specific certifications provided by OSVs.
    - Confusion from having distinct SR and ES certification bands.

Because of these factors, Arm has consolidated the SR and ES bands into a single compliance band, shifting from a one-time certification approach to a continuous, partner-driven compliance model. This new approach emphasizes ongoing integration and delivery, ensuring standards are maintained throughout a device's lifecycle, rather than at just one specific certification point. The compliance-

based model ultimately enhances product support and reduces troubleshooting over the entire lifespan of a device.

5.2 Will self-declared compliance be published on the Arm web page?

Arm publication of partners' compliance results has the same problems as certification, as described earlier. Therefore, Arm will not publish partner's self-declared compliance.

However, Arm is willing to participate in marketing promotions for a pathfinder. A pathfinder is a system on which both Arm and a partner expend SystemReady band pathfinding effort. For more information, see What is the SystemReady band pathfinding effort?.

5.3 What is the SystemReady band pathfinding effort?

The following points describe the SystemReady band pathfinding effort:

1. Arm works with selected strategic and willing partners to understand the SystemReady band compliance situation on silicon partners' reference platforms for new SoCs.
2. Criteria:
    - Strategic silicon partners proactively engage with Arm on the pathfinding effort.
    - Only new SoCs in development from the partners are considered. Existing SoCs already on the market are not eligible.
    - Only reference platforms from the silicon partners are considered.
    - Arm determines eligibility at its sole discretion, based on available resources.
3. Arm is willing to participate in marketing promotions for the pathfinder when its SystemReady band compliance is reviewed and confirmed by Arm.

5.4 How can a partner get support from Arm for the pathfinding effort?

To receive pathfinding support from Arm, confirm that your system meets the above criteria, then contact Arm using one of the following channels:

- The Arm SystemReady team
- Your Arm contact
- Submit a support request using the Contact Form


Note

If you only need technical support from Arm for pathfinding, without the need for co-marketing or the use of the SystemReady logo, it is not required to agree to the Arm SystemReady Trademark Agreement on the Contact Form.

5.5 Do partners need to work with Arm for self-declared compliance?

Working with Arm is not necessary.

5.6 How do partners complete the self-declared compliance process?

Partners independently run the test, review and evaluate the results, and then declare compliance.

When declaring compliance, partners can use the SystemReady band wordmark without Arm's permission. For example, partners can independently declare “our device XYZ is SystemReady Band v3.0 compliant”.

However, if partners also wish to use the SystemReady logo (that is, not just the wordmark) in their compliance declaration or want to co-market, partners must use the pathfinding process with Arm and sign the Arm SystemReady Trademark Agreement. In this case, please submit your request using the Contact Form.

For more information, see the Arm SystemReady Band Policy Guidelines.

5.7 How can partners independently evaluate test results and determine compliance?

Generally, partners (especially ODMs and OEMs) only need to focus on the failed tests (test failures) to determine SystemReady band compliance.

Partners may be able to ignore the following:

- Warnings from BSA and SBSA test suites
- Skipped tests
- Aborted tests
- Tests that generate warnings that are not critical for compliance.

More specifically:

- Warnings from BSA or SBSA tests are generally triggered by conditional requirements in the specifications that are based on platform specific information. Partners need to investigate using that platform specific information to determine whether a warning indicates an actual compliance issue.
- Skipped or not-run BSA and SBSA tests are likely already reviewed at the pre-silicon stage (before SoC tape-out) or at the post-silicon stage (during the reference system development) by silicon providers. Therefore, it may not be necessary for system vendors to recheck them.
- For FWTS and SCT, the test results marked as skipped, aborted, or warnings are generally not critical for determining compliance. These results are typically either suggestions for improvement or expected test results based on conditional, optional, or recommended requirements in the BBR specification.
- For SBMR In-band, the test results marked as skipped are generally conditional requirements based on platform-specific information. Partners need to investigate using that platform-specific information to determine whether a skipped test indicates an actual compliance issue.

5.8 How can partners decide whether a test failure is critical for self-compliance?

Partners can use the following steps to filter out non-critical failed tests:

1. Run the compliance testing on a reference board and check if the failures with your system are the same as the ones with the reference board. If so, the failures might not be critical. It is recommended to consult with the SoC vendor to confirm whether these failures are non-critical. Note that if partners want to check compliance on a reference board with Arm, please refer to the previous question about pathfinding effort.
2. Remove all PCIe cards and re-run the testing to check if the failure is with a PCIe card. If so, this failure might not be critical for compliance. Partner can either replace the problematic PCIe card or refer to Potentially waivable failures for SystemReady band compliance for more information about the non-critical failures.
3. Check the previous certification waiver document or previous compliance report.
4. Check the Potentially waivable failures for SystemReady band compliance in this document.

After partners filter out the non-critical failures, there should be no hardware compliance (BSA/SBSA) failure remaining in determining compliance. Any remaining failures should be only from SCT or FWTS tests, which are system firmware issues. In this case, partners should work with the system firmware team, IBV (Independent BIOS Vendor), or IFV (Independent Firmware Vendor) to fix the issues or to seek clarification or a waiver justification.

For more information on reviewing the compliance test result and determining the compliance, please refer to the SystemReady Band Integration and Testing Guide.


Note

If partners believe the failure is a test suite issue or a specification issue and require clarification and support from Arm, they are welcome to raise an issue on the arm-systemready GitHub repo for a test suite issue, or a SystemArchAC mantis ticket for a specification issue.

5.9 Can partners use a beta version of an ACS live image (for example v24.11_3.0.0-BET0) for self-declared compliance?

Yes. It is acceptable to use a beta version of an ACS live image, as long as the ACS version is mentioned in the version of the SRS specification that you are declaring compliance with.

SRS v3.0 allows v24.11_3.0.0-BET0 and later versions for compliance testing. This means that if you want to declare your system is SystemReady band v3.0 compliant, you can use any ACS images on Release pre-build images including the future releases for compliance testing.

5.10 How often is the ACS live image released?

The ACS live image is released twice a year. However, if there is a critical issue with the latest release, or a critical need arises, an additional release may occur. For more information about releases, please send an email to support-systemready-acs@arm.com.

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

For SystemReady band compliance, the Arm Neoverse reference design platforms are available from the following location:

- Infrastructure FVPs
- Arm Neoverse reference Design Platform Software

For more information about how to run the ACS tests as part of a SystemReady live image, refer to the SystemReady Band Integration and Testing Guide. In particular, the Set up the RD-N2 FVP, Set up the Radxa Orion O6, and ACS sections provide an examples of how to set up a system to run the compliance testing.

6. SystemReady compliance testing - Pre-silicon FAQ

This section provides answers to questions about compliance testing for pre-silicon enablement.

6.1 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.

6.2 What are common BSA/SBSA 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.

6.3 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. 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 BSA/SBSA Integration and Testing Guide for a comprehensive list of common compliance issues, and information about how to address and prevent them.

6.4 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 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.

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

The compliance test suites 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 compliance, on-silicon BSA/SBSA 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.

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

The BSA and SBSA 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.

6.7 Should I run the pre-silicon tests on UEFI or bare metal?

You can run the BSA/SBSA 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.

6.8 How long does it take to port an ACS suite to my bare metal platform?

Porting an Architecture Compliance Suite (ACS) to a bare metal system involves several crucial steps:

1. Integrating ACS into the Bare Metal Boot Flow: This entails seamlessly incorporating ACS into the boot sequence of the bare metal system to ensure it runs efficiently from startup.
2. Providing Platform-Specific Information: Tailoring ACS to the specific hardware configuration of the platform by supplying detailed platform-specific information, ensuring optimal compatibility.
3. Implementing Platform-Specific PAL (Platform Abstraction Layer) APIs: Developing the necessary PAL APIs specific to the platform, facilitating communication between ACS and the underlying hardware components.

For comprehensive guidance on this porting process, refer to the detailed instructions available at Arm SBSA Architecture Compliance Bare Metal User Guide.

In contrast, for UEFI-based systems, the porting effort is significantly streamlined. Only the exerciser PAL APIs need to be implemented, allowing for the compilation of ACS as a UEFI App. This approach eliminates the need for extensive customization efforts.

For detailed instructions on porting the exerciser PAL APIs, refer to the documentation provided at Exerciser API Porting Guide.

6.9 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.

6.10 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.

7. SystemReady compliance testing - Exerciser FAQ

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

7.1 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

7.2 How does the exerciser integrate with the test environment?

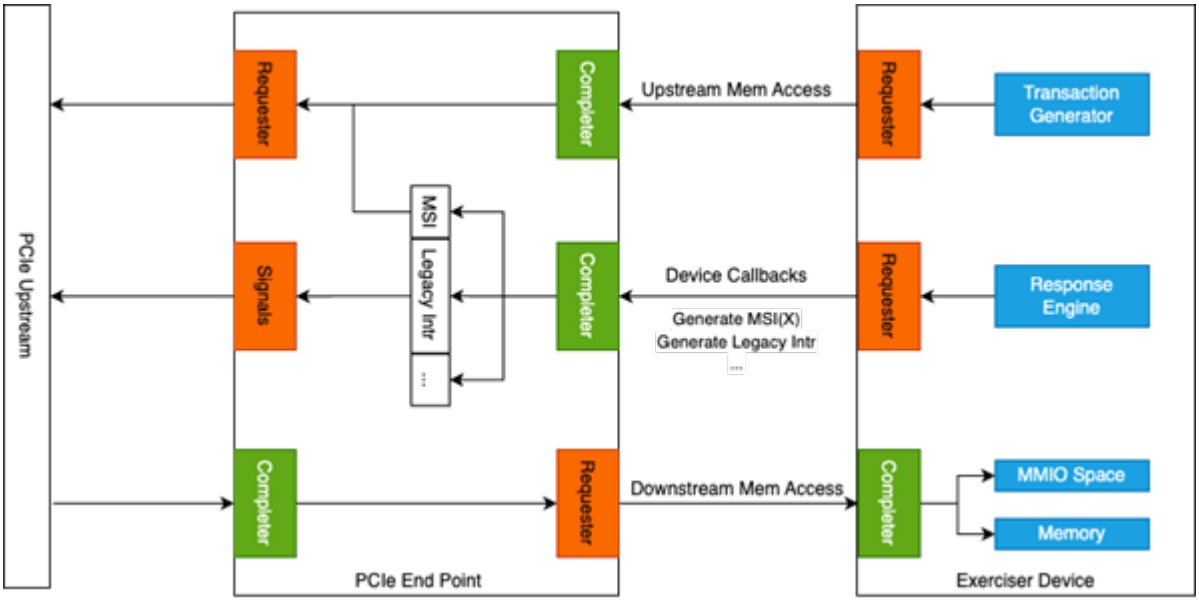
Validating the compliance of specific PCIe rules defined in the BSA/SBSA specification requires the PCIe endpoint to generate specific stimuli during the runtime of the test. Examples of such stimuli

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 7-1: Block diagram for exerciser PCIe endpoint



7.3 How does the exerciser integrate with an Arm SoC?

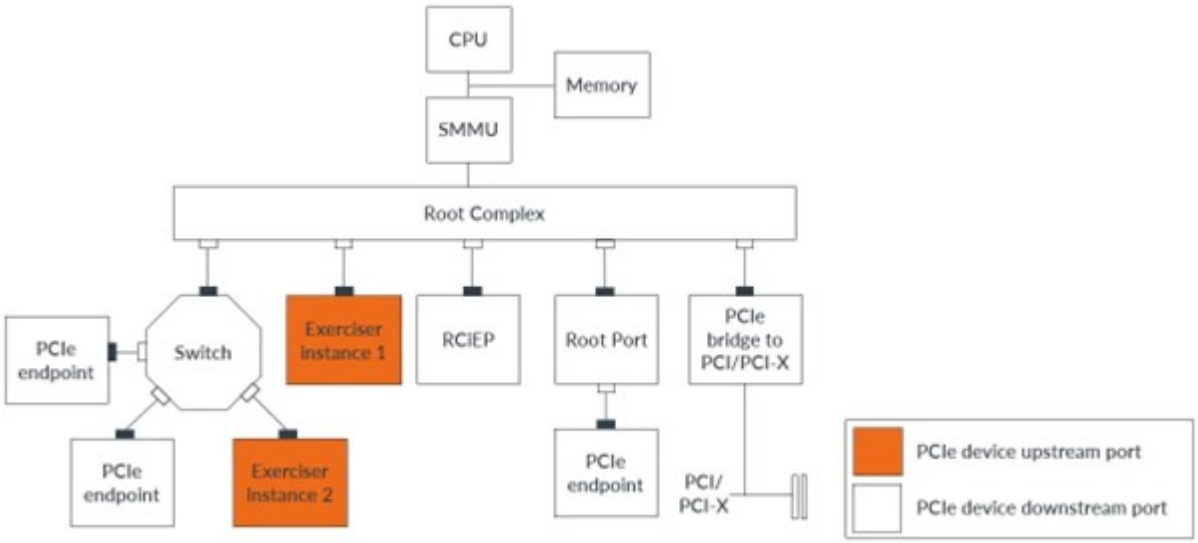
The following block diagram shows two instances of exerciser that are present in the system:

- 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 7-2: Block diagram for exerciser representation in a system



7.4 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?.

7.5 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.

7.6 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.

7.7 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.

8. SystemReady compliance testing - BBSR and Security Interface Testing FAQ

This section provides answers to questions about compliance testing for the BBSR and security interface.

8.1 What is BBSR?

The Base Boot Security Requirements (BBSR) specification describes the security requirements for a device to comply with industry-standard security interfaces and covers the following areas:

- UEFI authenticated variables
- UEFI secure boot
- UEFI capsule updates
- TPM 2.0 and measured boot

However, interface compliance does not provide assurance that the underlying platform is secure. When architecting a system, system-level threat modeling should be performed to evaluate threats, risks, and mitigations.

For example, in the embedded IoT market, the Platform Security Architecture (PSA) and the PSA Certified framework provides a comprehensive approach to platform security that is based on a defined set of security goals. PSA provides architecture and requirements specifications for building secure platforms. BBSR compliance complements PSA. PSA Certified shows the robustness of an implementation, through an assessment process that is performed by a security compliance laboratory.

8.2 How do I test for Base Boot Security Requirements (BBSR) compliance?

The BBSR ACS is a compliance suite that tests for compliance with the requirements specified in the Base Boot Security Requirements specification. The BBSR-ACS provides assurance that the security interfaces covered by the BBSR are implemented according to specified standards.

The BBSR ACS tests are included as part of the SystemReady ACS live image that is available for each band of SystemReady. Pre-built ACS images for each band of SystemReady are available on Github: https://github.com/ARM-software/arm-systemready.

This live image is deployed onto a storage device, such as a USB drive, which can then be used on the system under test. Please follow the deployment steps specified in the integration guide appropriate to the SystemReady band being tested.

The following tests must pass to achieve BBSR compliance:

- SCT authenticated variable tests
- SCT secure boot variable size test
- SCT secure boot image loading, variable update, and variable attribute tests
- FWTS authenticated variable tests
- Firmware update using an update capsule

If a TPM is present on the system and the system supports TPM measured boot, the following additional tests must pass:

- SCT TCG2 protocol test
- FWTS tpm2 test
- Evaluation of the measured boot log

8.2.1 Enroll the Secure Boot keys

The SystemReady ACS live image provides a set of keys for the UEFI Secure Boot variables PK, KEK, db, and dbx. Before running the test suite these test keys must be enrolled using the platform-specific procedure for the firmware of the platform under test.

The test keys are available on the `boot` partition of the ACS image at the following path:

```
acs_tests/bbsr-keys/
```

The test keys are available in the following formats:

- DER format, suitable for enrolling in EDK2
- Signed UEFI variable signature list blobs, suitable for U-boot

The following files are the DER formatted test keys:

- `TestDB1.der`
- `TestDBX1.der`
- `TestKEK1.der`
- `TestPK1.der`

The following files are the signed signature list formatted test keys:

- `TestDB1.auth`
- `TestDBX1.auth`
- `TestKEK1.auth`
- `TestPK1.auth`

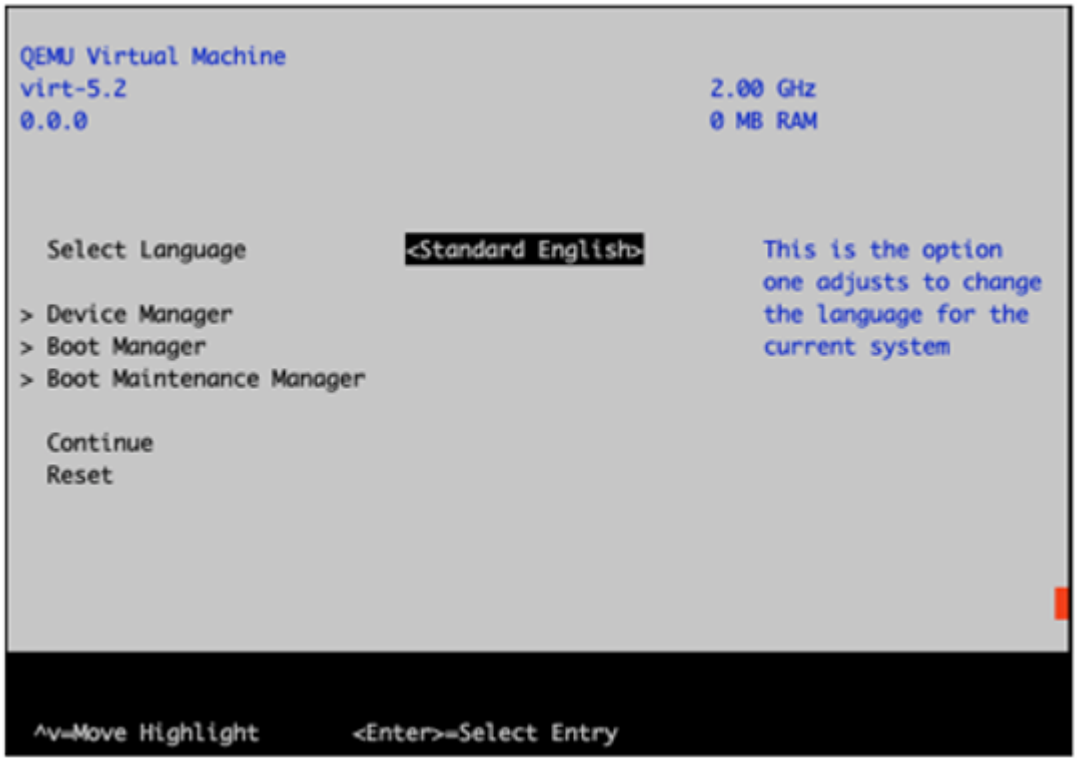
8.2.1.1 Example: Enrolling keys in EDK2

The example in this section shows how to enroll the Secure Boot keys on QEMU with EDK2-based firmware.

Perform the following steps:

1. After starting the system, press ESC to enter the EDK2 menu.

Figure 8-1: EDK2 menu example



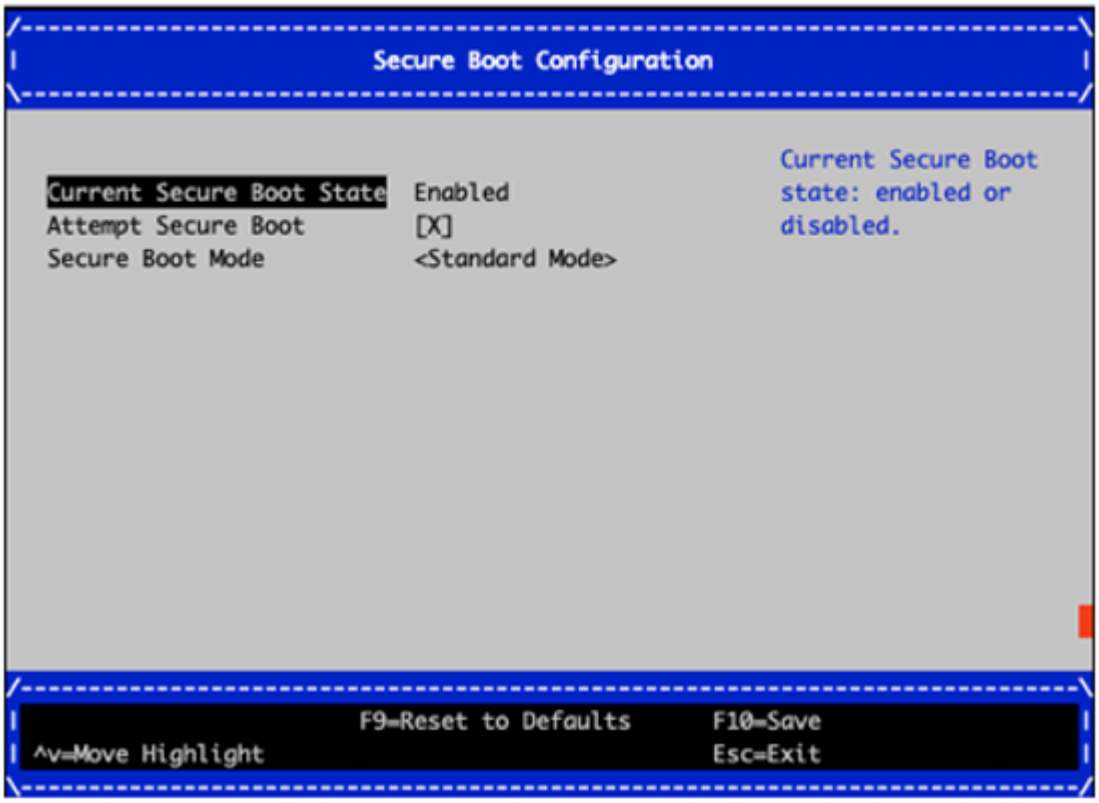
2. Select Device Manager > Secure Boot Configuration.
3. Select Custom Mode for Secure Boot Mode.
4. Select Custom Secure Boot Options.
5. To enroll the Platform Key (PK), do the following:
   a. Select PK Options.
   b. Select Enroll PK > Enroll PK Using File.
   c. Select the SystemReady ACS disk which has the BOOT label.
   d. The secure boot keys are located at the following path on the disk:
      - `acs_tests/bbsr-keys`
   e. Select the `TestPK1.der` file for PK.
   f. Commit the changes.

g. Repeat the above steps to enroll the keys for KEK (`TestKEK1.der`), db (`TestDB1.der`), and dbx (`TestDBX1.der`) selecting the following options:

- KEK Options
- DB Options
- DBX Options

After completing the above steps, secure boot is enabled as the following diagram shows:

Figure 8-2: Secure Boot Configuration Example



6. Reset the system.

8.2.1.2 Example: Enrolling keys in U-boot

Note: The enrollment of Secure Boot keys in U-Boot-based systems has been automated as part of the ACS (Architectural Compliance Suite) process. The following steps should only be used if the automated enrollment fails or for troubleshooting purposes.

For information about enrolling keys with U-boot firmware, see the U-boot document UEFI on U-Boot.

For U-boot, the method to access and load the Secure Boot keys differs depending on the type of physical storage device where the SystemReady ACS is located.

For example, to initialize the USB subsystem:

```
==> usb start
```

To initialize MMC device 1:

```
==> mmc dev 1
```

For more details see the U-boot documentation for the storage device type in use.

The following example shows how to enroll PK, KEK, db, and dbx with the SystemReady ACS provided keys loaded from USB device 0:

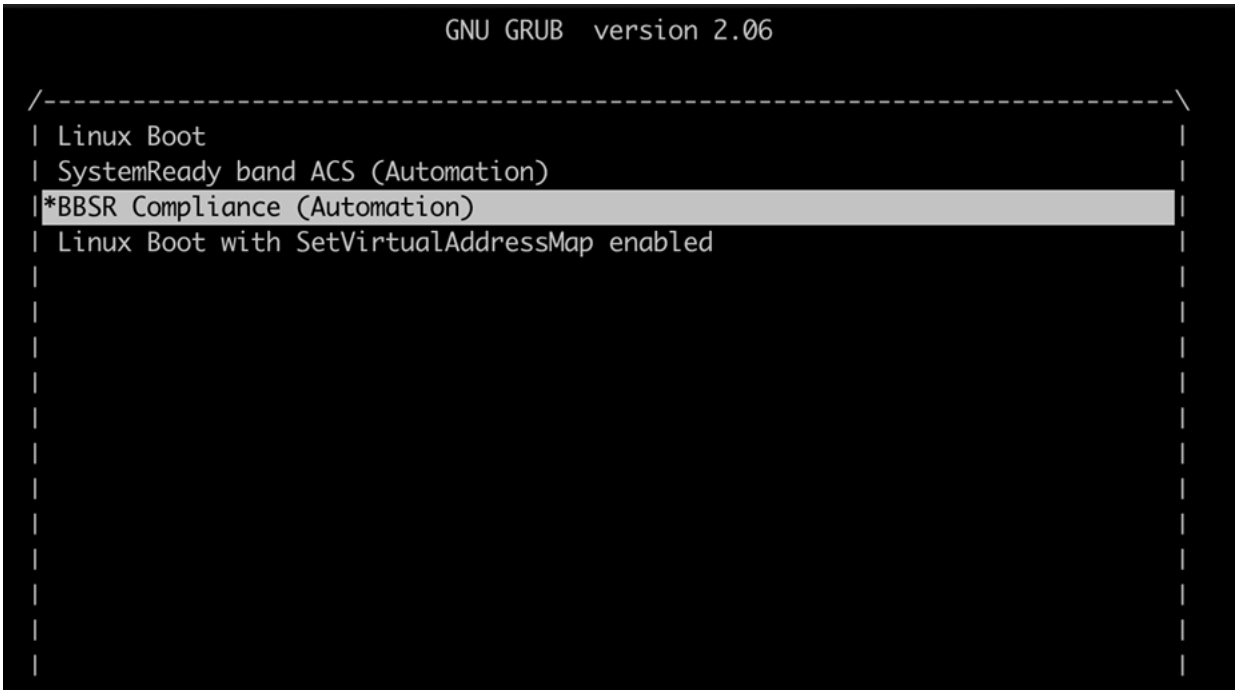
```
==> load usb 0 ${loadaddr} acs_tests/bbsr-keys/TestPK1.auth && setenv -e -nv -bs -rt
 -at -i ${loadaddr}:$filesize PK
==> load usb 0 ${loadaddr} acs_tests/bbsr-keys/TestKEK1.auth && setenv -e -nv -bs -
rt -at -i ${loadaddr}:$filesize KEK
==> load usb 0 ${loadaddr} acs_tests/bbsr-keys/TestDB1.auth && setenv -e -nv -bs -rt
 -at -i ${loadaddr}:$filesize db
==> load usb 0 ${loadaddr} acs_tests/bbsr-keys/TestDBX1.auth && setenv -e -nv -bs -
rt -at -i ${loadaddr}:$filesize dbx
```

8.2.2 BBSR SCT tests

The Base Boot Security Requirements SCT is a subset of the main SCT test suite and tests security interfaces: authenticated variables, Secure Boot variables, Secure Boot image loading, and TCG2 protocol test for systems with TPMs.

After resetting the system with the SystemReady ACS Secure Boot keys enrolled, select the BBSR Compliance (Automation) option from the GRUB menu within 5 seconds to start the Base Boot Security Requirements SCT.

Figure 8-3: GRUB menu



During the SCT run, the system may restart multiple times. BBSR SCT automatically resumes ; there is no need to interrupt or make a GRUB menu selection each time. After SCT completes, the system boots to Linux in Secure Boot mode to continue FWTS in the Linux environment and perform TPM log analysis if a TPM is present in the system.

The BBSR SCT test output is available in the `acs_results\BBSR` directory of the ACS disk:

- `\acs_results\BBSR\sct_results\Overall\Summary.log`

Note

If a TPM is present in the system and is supported by the firmware, the TCG2 protocol tests in SCT run and are included in the `Summary.log` file. If the TCG2 protocol is not present, these tests do not run.

8.2.3 BBSR FWTS tests

FWTS is a set of Linux-based firmware tests. The BBSR ACS runs a subset of FWTS: the authenticated variable tests and `tpm2`, the Trusted Platform Module 2 test. Also, if the system under test implements a TPM device, commands are executed to capture the results of TPM measured boot.

After SCT completes, the system boots to Linux in Secure Boot mode to continue FWTS in the Linux environment and perform TPM log analysis if a TPM is present in the system. Once the system boots to Linux, it automatically runs a subset of FWTS and outputs the TPM event log and PCRs.

If no TPM is present, the `tpm2` test and output of PCRs and the event log is skipped:

```
Test Executed are uefirtauthvar tpm2
Running 2 tests, results appended to /mnt/acs_results/fwts/FWTSResults.log
Test: Authenticated variable tests.
  Create authenticated variable test.                    1 passed
  Authenticated variable test with the same authentica..  1 passed
  Authenticated variable test with another valid authe..  1 passed
  Append authenticated variable test.                    1 passed
  Update authenticated variable test.                    1 passed
  Authenticated variable test with old authenticated v..  1 passed
  Delete authenticated variable test.                    1 passed
  Authenticated variable test with invalid modified data  1 passed
  Authenticated variable test with invalid modified ti..  1 passed
  Authenticated variable test with different guid.        1 passed
  Set and delete authenticated variable created by dif..  2 passed
Test: TPM2 Trusted Platform Module 2 test.
 Test skipped.
TPM event log not found at /sys/kernel/security/tpm0/binary_bios_measurements
```

If a TPM is present, the `tpm2` test is run and PCRs and event log are output to the `acs_results` directory:

```
Test Executed are uefirtauthvar tpm2
Running 2 tests, results appended to /mnt/acs_results/fwts/FWTSResults.log
Test: Authenticated variable tests.
  Create authenticated variable test.                    1 passed
  Authenticated variable test with the same authentica..  1 passed
  Authenticated variable test with another valid authe..  1 passed
  Append authenticated variable test.                    1 passed
  Update authenticated variable test.                    1 passed
  Authenticated variable test with old authenticated v..  1 passed
  Delete authenticated variable test.                    1 passed
  Authenticated variable test with invalid modified data  1 passed
  Authenticated variable test with invalid modified ti..  1 passed
  Authenticated variable test with different guid.        1 passed
  Set and delete authenticated variable created by dif..  2 passed
Test: TPM2 Trusted Platform Module 2 test.
  Validate TPM2 table.                                   1 passed
TPM2: dumping PCRs and event log
  Event log: /mnt/acs_results/tmp2/eventlog.log
  PCRs: /mnt/acs_results/tmp2/pcr.log
```

You can see the test logs in the `BOOT_ACS` partition of the SystemReady ACS storage device:

- FWTS results: `acs_results/BBSR/fwts/FWTSResults.log`
- Event log: `acs_results/BBSR/tpm2/eventlog.log`
- PCRs: `acs_results/BBSR/tpm2/pcr.log`

8.2.4 Secure firmware update test

If the system only supports in-band firmware updates, the BBSR specification requires support for update capsules compliant with the UEFI specification for systems that perform in-band firmware updates. The BBSR ACS firmware update test is a manual test run from the firmware that requires a valid update capsule for the system's firmware.

The UEFI specification defines two ways to perform updates with capsules:

- The `UpdateCapsule()` runtime function (see UEFI section 8.5.3)
- Capsule on disk (see UEFI section 8.5.5)

Both ways are acceptable for performing the capsule update test. The vendor of the System Under Test (SUT) must provide the update procedure to use.

Some of the steps in the test procedure below use the `CapsuleApp.efi` program. This in the SystemReady ACS image at the following path: `EFI\BOOT\app\CapsuleApp.efi`.

Step #1. Preparation

Perform the following steps to prepare for the firmware update test:

1. Copy the vendor-provided update capsule image onto a storage device.
2. Prepare an invalid copy of the vendor-provided update capsule that has been tampered with to test the firmware's ability to reject invalid capsules. Using a copy of the vendor-provided update capsule, use a binary editor such as the `xxd` command and an editor to manually modify the last byte of the image.
3. Copy the tampered update capsule onto the storage device.
4. Enable or install the storage device containing the capsule images on the SUT so that it is visible to firmware.

Step #2. Reset the system and get to the UEFI Shell

The firmware update tests are run manually from the UEFI Shell. Reset the system and from the UEFI Shell change to the ACS results partition and create a directory named `fwupdate`. In the example below the results partition is on the `FS1` drive.

```
FS0:\> fs1:
FS1:\> cd acs_results\BBSR
FS1:\acs_results\BBSR\> mkdir fwupdate
FS1:\acs_results\BBSR\> cd fwupdate
FS1:\acs_results\BBSR\fwupdate\>
```

Step #3. Dump the ESRT

From the `acs_results\BBSR\fwupdate` directory use the `CapsuleApp -E` command to dump the firmware's EFI System Resource Table (ESRT) into a log file named `esrt_dump.log`, as follows:

```
FS1:\acs_results\BBSR\fwupdate\> CapsuleApp -E > esrt_dump.log
```

Step #4. Dump the FMP information advertised by the firmware

From the `acs_results\BBSR\fwupdate` directory use the `CapsuleApp -P` command to dump the Firmware Management Protocol (FMP) information into a log file named `fmp_dump.log`, as follows:

```
FS1:\acs_results\BBSR\fwupdate\> CapsuleApp -P > fmp_dump.log
```

Step #5. Dump the update capsule header

From the `acs_results\BBSR\fwupdate` directory use the `CapsuleApp -D` command to dump the header of the vendor-provided update capsule into a log file named `capsule_header.log`, as follows:

```
FS1:\acs_results\BBSR\fwupdate\> CapsuleApp -D [capsule-image] > capsule_header.log
```

Step #6. Test a firmware update with the tampered capsule image

This is a negative test that verifies whether the firmware update process correctly rejects a capsule that has been tampered with. The expected result is that the firmware update must not be processed.

To perform this test, follow the vendor-provided firmware update procedure to perform the update using a capsule that has been tampered with, as described in Step #1. Preparation.

For example, a firmware update using the `CapsuleApp.efi` utility might look like this:

```
FS1:\acs_results\BBSR\fwupdate\> CapsuleApp FS2:\tampered.bin >
 fwupdate_tampered.log
```

Step #7. Test a firmware update using the CapsuleApp with the vendor-provided capsule

This test step verifies that the vendor-provided update capsule is applied correctly. To perform this test, follow these steps:

1. Use the vendor-provided procedure to perform the update using the valid vendor-provided update capsule. The expected result is that the capsule is processed successfully, and the firmware is updated.

   For example, a firmware update using the CapsuleApp.efi utility might look like this:

   ```
   FS1:\acs_results\BBSR\fwupdate\> CapsuleApp FS2:\DeveloperBox.cap > fwupdate.log
   ```

2. Reset the system.
3. Repeat the test step to dump the FMP into a log file named `fmp_post_update_dump.log` to verify that the FMP advertises the new firmware version:

   ```
   FS1:\acs_results\BBSR\fwupdate\> CapsuleApp -P > fmp_post_update_dump.log
   ```

8.2.5 Review the ACS test result logs

The log files generated by running the BBSR ACS tests are in a separate directory within the live image called `acs_results/BBSR`.

You can see the results partition at the directory `/mnt` when booted into the ACS Linux on the system under test.

You can also view the storage device holding the results partition on a host machine.

8.2.5.1 Review the SCT logs

Review the SCT summary log created during the SCT phase of test execution at the following location:

```
acs_results/BBSR/sct_results/Overall/Summary.log
```

The expected result is that the following tests pass:

- `RuntimeServicesTest\VariableServicesTest\AuthVar_Conf` (34 tests)
- `RuntimeServicesTest\VariableServicesTest\AuthVar_Func` (16 tests)
- `RuntimeServicesTest\BBSRVariableSizeTest\BBSRVariableSizeTest_func` (2 tests)
- `RuntimeServicesTest\SecureBootTest\ImageLoading` (20 tests)
- `RuntimeServicesTest\SecureBootTest\VariableAttributes` (12 tests)
- `RuntimeServicesTest\SecureBootTest\VariableUpdates` (10 tests)
- `RuntimeServicesTest\TCGMemoryOverwriteRequestTest\Test MOR and MORLOCK` (49 tests)
- `GenericTest\PlatformResetAttackMitigationPsciTest\` (1 test)

If a TPM is present in the system and is supported by the firmware, the TCG2 protocol tests in SCT run and are included in the `Summary.log` file. If the TCG2 protocol is not present, these tests do not run. The following TCG2 protocol tests must pass:

- `TCG2ProtocolTest\GetActivePcrBanks_Conf` (2 tests)
- `TCG2ProtocolTest\GetCapability_Conf` (3 tests)
- `TCG2ProtocolTest\HashLogExtendEvent_Conf` (10 tests)
- `TCG2ProtocolTest\SubmitCommand_Conf` (2 tests)

8.2.5.2 Review the FWTS logs

Review the FWTS results log created during the FWTS phase of test execution at the following location:

```
acs_results/BBSR/fwts/FWTSResults.log
```

The expected result is that all tests pass. The `tpm2` test is only applicable to ACPI-based systems that support TPM measured boot. An example of the expected results is shown below:

```
Test: Authenticated variable tests.
  Create authenticated variable test.                    1 passed
  Authenticated variable test with the same authentica..  1 passed
  Authenticated variable test with another valid authe..  1 passed
  Append authenticated variable test.                    1 passed
```

```
  Update authenticated variable test.                    1 passed
  Authenticated variable test with old authenticated v..  1 passed
  Delete authenticated variable test.                    1 passed
  Authenticated variable test with invalid modified data  1 passed
  Authenticated variable test with invalid modified ti..  1 passed
  Authenticated variable test with different guid.        1 passed
  Set and delete authenticated variable created by dif..  2 passed
Test: TPM2 Trusted Platform Module 2 test.
  Validate TPM2 table.                                   1 passed
```

8.2.5.3 Review the firmware update logs

You must review the following logs created during the firmware update test procedure:

- ESRT dump: `fwupdate/esrt_dump.log`
- FMP dump: `fwupdate/fmp_dump.log`
- Capsule header dump: `fwupdate/capsule_header.log`
- Tampered firmware update log: `fwupdate/fwupdate_tampered.log`
- Firmware update log: `fwupdate/fwupdate.log`
- Post-firmware update FMP dump: `fwupdate/fmp_post_update_dump.log`

8.2.5.4 Review the TPM measured boot log

The TPM log review process has been automated using a Python script, eliminating the need for manual analysis of TPM logs generated during the FWTS phase of test execution. The script automatically verifies compliance against the TCG Firmware Profile specification and generates a summary report as follows:

- Event log: `acs_results/BBSR/tpm2/eventlog.log`
- PCRs log: `acs_results/BBSR/tpm2/pcr.log`
- Compliance results: `acs_results/BBSR/tpm2/verify_tpm_measurements.log`

The script ensures structured validation, as TPM measurements are highly platform-specific. This documentation now serves as a reference for the script's functionality rather than outlining manual review steps.

The steps below show how to verify key requirements defined in the TCG Firmware Profile specification. The measurements for a particular system are highly platform-specific. The TCG Firmware Profile specification dictates the specific requirements.

1. Verify that the cumulative SHA256 measurements from the event log match the TPM PCRs 0-7.

   The events logged in the TPM event log must match the actual measurements extended in the TPM PCRs. You can perform a visual comparison of this by viewing the SHA256 PCR values in the `pcr.log` file and the computed values at the end of `eventlog.log`.

The following example shows where the PCR values and event log values match.

```
SHA256 values for PCRs 0-7 from pcr.log
sha256:
  0 : 0x4A17B720C5E37DCD65533EB47CDE5B5E1E93E9A5953B42E913F2C83D88576685
  1 : 0x8EFBC5102BEB859074EC99DB20009BD213726B57777DA560B7BC7AA567C22425
  2 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  3 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  4 : 0xFE3C30CA8D4CACCAAAE635D60DC3132D1B5C93E0F2BB092BF0D83287D76B1210
  5 : 0x768A45048228ECE6EF442FA88AF60DFB19D8ABCB1869E1DBDBEAEA1244353037
  6 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  7 : 0x7D852DB48CA55F36243903877E416D4AF77AA8755010C064884799E70F51664D

SHA256 values for PCRs 0-7 from eventlog.log
pcrs:
  sha256:
    0  : 0x4a17b720c5e37dcd65533eb47cde5b5e1e93e9a5953b42e913f2c83d88576685
    1  : 0x8efbc5102beb859074ec99db20009bd213726b57777da560b7bc7aa567c22425
    2  : 0x3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    3  : 0x3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    4  : 0xfe3c30ca8d4caccaaae635d60dc3132d1b5c93e0f2bb092bf0d83287d76b1210
    5  : 0x768a45048228ece6ef442fa88af60dfb19d8abcb1869e1dbdbeaea1244353037
    6  : 0x3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    7  : 0x7d852db48ca55f36243903877e416d4af77aa8755010c064884799e70f51664d
```

2. Verify the `EV_NO_ACTION` event for Specification ID version.

   The first event in the event log must be the Specification ID version. This is an `EV_NO_ACTION` event and is not extended into a PCR. For example:

```
- EventNum: 0
  PCRIndex: 0
  EventType: EV_NO_ACTION
  Digest: "0000000000000000000000000000000000000000"
  EventSize: 45
  SpecID:
  - Signature: Spec ID Event03
    platformClass: 0
    specVersionMinor: 0
    specVersionMajor: 2
    specErrata: 0
    uintnSize: 2
    numberOfAlgorithms: 4
```

3. Verify `EV_POST_CODE` events for measurements of firmware to PCR[0].

   All mutable Secure and Non-secure firmware components must be measured into PCR[0] using the `EV_POST_CODE` event type. BBSR provides the suggested event data values.

4. Verify `EV_POST_CODE` events for measurements of signed critical to data PCR[0].

   All signed critical data must be measured into PCR[0] using the `EV_POST_CODE` event type, with platform-specific event data.

5. Verify Secure Boot policy measurements.

   The contents of the `SecureBoot`, `PK`, `KEK`, `db`, and `dbx` variables must be measured into PCR[7] using the `EV_EFI_VARIABLE_DRIVER_CONFIG` event type.

The following example shows the measurement of the SecureBoot variable:

```
- EventNum: 3
  PCRIndex: 7
  EventType: EV_EFI_VARIABLE_DRIVER_CONFIG
  DigestCount: 4
  Digests:
  - AlgorithmId: sha1
    Digest: "d4fdd1f14d4041494deb8fc990c45343d2277d08"
  - AlgorithmId: sha256
    Digest: "ccfc4bb32888a345bc8aeadaba552b627d99348c767681ab3141f5b01e40a40e"
  - AlgorithmId: sha384
    Digest:
 "2cded0c6f453d4c6f59c5e14ec61abc6b018314540a2367cba326a52aa2b315ccc08ce68a816ce09c6ef2ac7e51
  - AlgorithmId: sha512
    Digest:
 "94a377e9002be6e1d8399bf7674d9eb4e931df34f48709fddd5e1493bfb96c19ee695387109a5a5b42f4871cbee
  EventSize: 53
  Event:
    VariableName: 61dfe48b-ca93-d211-aa0d-00e098032b8c
    UnicodeNameLength: 10
    VariableDataLength: 1
    UnicodeName: SecureBoot
    VariableData: "01"
```

6. UEFI BootOrder and Boot#### variables.

   If the UEFI `BootOrder` and `Boot####` variables are used by the firmware, they must be measured into PCR[1] with event types `EV_EFI_VARIABLE_BOOT` or `EV_EFI_VARIABLE_BOOT2`.

7. Verify boot attempt measurements

   Platform firmware must record each boot attempt into PCR[4] using the event type `EV_EFI_ACTION` with the action string `"Calling EFI Application from Boot Option"`.

8. Verify PCR[1] measurements.

   Measurements of security relevant configuration data go into PCR[1]. This should include configuration data such as the security lifecycle state of a system.

   Security relevant SMBIOS structures must be measured into PCR[1] using event type `EV_EFI_HANDOFF_TABLES`. This should include structures that identify the platform hardware for example manufacturer, model number, version, and so on.

9. Verify `EV_SEPARATOR` measurements

   The `EV_SEPARATOR` event delineates the point in platform boot where the platform firmware relinquishes control of making measurements into the TPM. There must be an `EV_SEPARATOR` measurement for each PCR[0] through PCR[7].

9. SystemReady compliance testing – SBMR FAQ

This section provides answers to general questions about compliance testing for the SBMR.

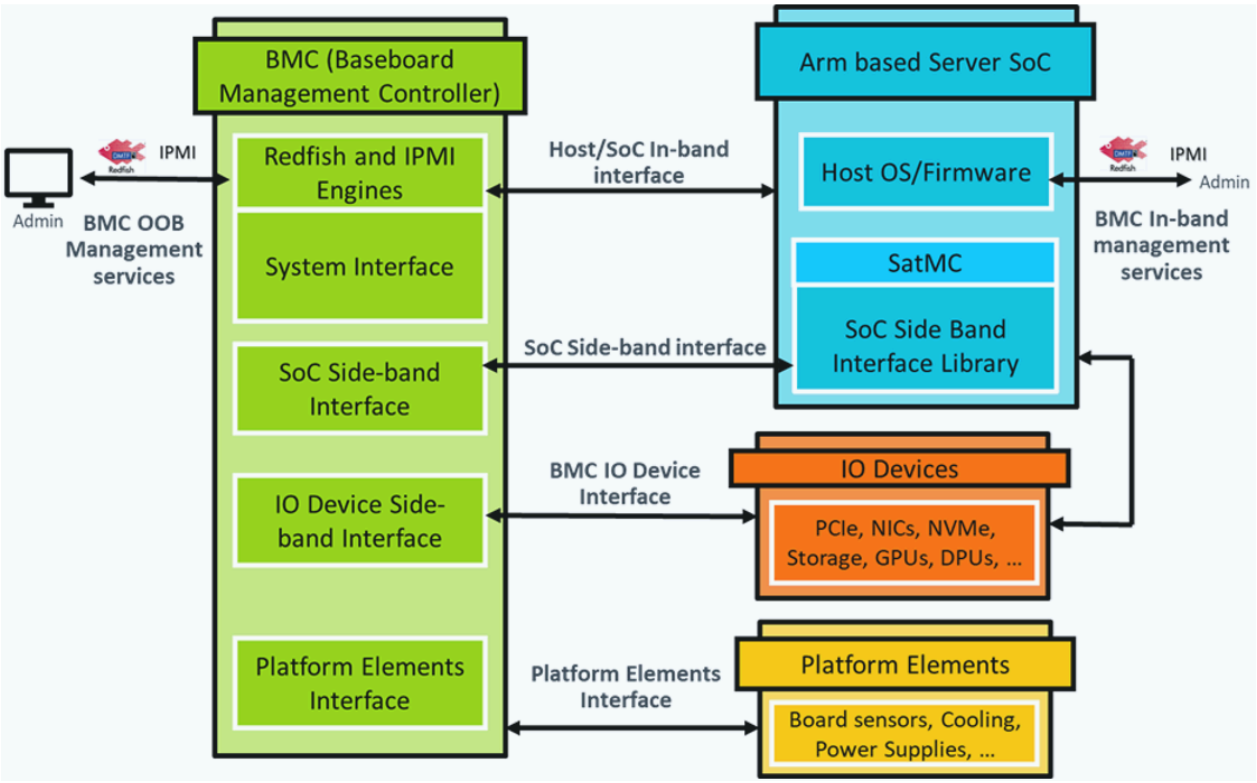
9.1 What is SBMR?

Arm Server Base Manageability Requirements (SBMR) specification describes a standardized set of requirements for manageability on Arm AArch64 servers. SBMR establishes a baseline for essential interfaces and functionalities to ensure interoperability across Arm servers. It standardizes methods for monitoring, firmware updates, remote management, telemetry, and debugging workflows.
By aligning with industry standards such as Redfish, IPMI, PLDM, and MCTP, SBMR enhances compatibility across hardware vendors and datacenter environments.

The document addresses the common standard interface sets as illustrated below:

- Arm SoC-BMC interface
  - Host SoC in-band interface
  - SoC side-band interface
- BMC-IO device interface
- BMC-Platform elements interface
- BMC Out-of-Band (OOB) remote management interface

Figure 9-1: SBMR Standard Interfaces



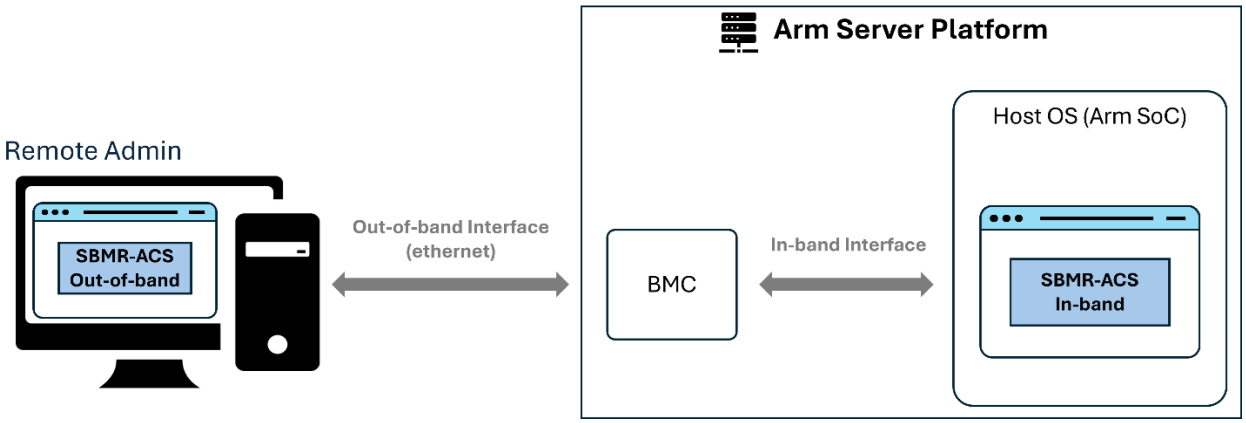
9.2 How do I test for Server Base Manageability Requirements?

SBMR Architecture Compliance Suite (ACS) is an open-source test suite based on the Robot Framework, designed to verify that server implementations conform to the SBMR specification. It supports testing of IPMI, Redfish, Host Interfaces and so on.

SBMR-ACS consists of two components: the out-of-band test suite and the in-band test suite. Both must be executed to ensure full compliance with the SBMR specification.

- The out-of-band test suite should be run from an external host machine running a Linux distribution.
- The in-band test suite should be executed directly on the system under test, also running a Linux distribution.

Figure 9-2: SBMR-ACS Setup Block Diagram



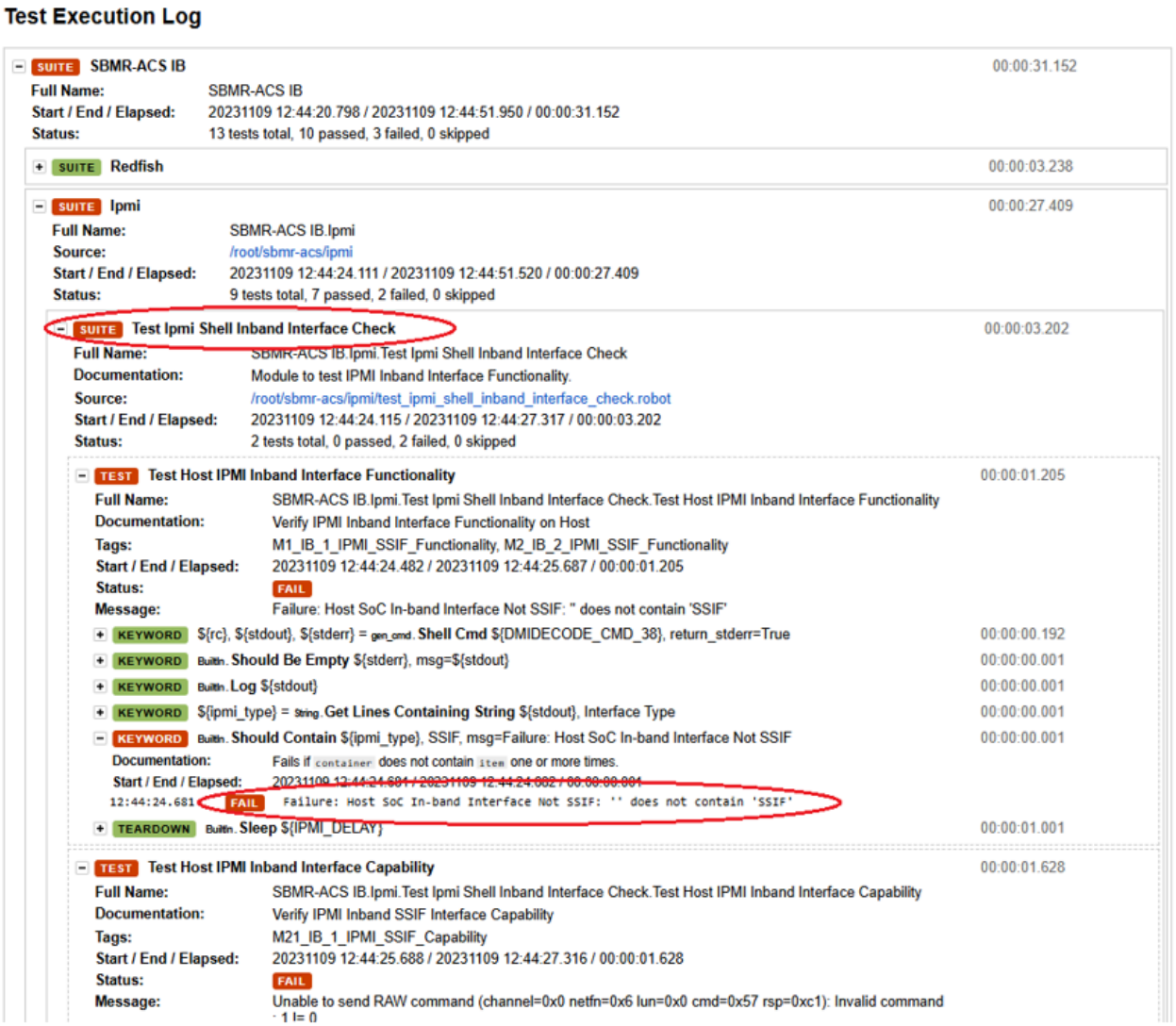
SBMR In-band test suite is integrated into the SystemReady Band ACS live image and can be executed by launching the ACS automated test run. For more details, refer to the ACS chapter in the Arm SystemReady Band Integration and Testing Guide.

SBMR Out-of-Band test suite must be executed separately to achieve full compliance with the SBMR specification. For more details, refer to the SBMR-ACS GitHub repository.

To review the test results, open `log.html` in a web browser from the `sbmr-acs` directory. Successful test suites are highlighted in green, and failed test suites are highlighted in red for easy identification. Additional details are provided in `log.html`.

Test suites marked as `SKIP` and highlighted in yellow are generally conditional requirements that depend on platform-specific information. Partners should review using that platform-specific information to determine whether a skipped test indicates an actual compliance issue.

Figure 9-3: SBMR-ACS In-band Test Result Example



10. SystemReady Devicetree band – General FAQ

This section provides answers to general questions about Arm SystemReady Devicetree.

10.1 What operating systems can run on a SystemReady Devicetree platform?

While SystemReady Devicetree is intended to make it easier to build embedded Linux and BSD systems, it defines a base platform architecture that can be used by any operating system. Operating systems that use the UEFI firmware ABI and the Devicetree system description can boot on a SystemReady Devicetree platform.

10.2 How does SystemReady Devicetree band differ from SystemReady band?

SystemReady Devicetree band differs from SystemReady band in the following ways:

- SystemReady Devicetree requires only a subset of the UEFI ABI required by the SystemReady core band. In particular, SystemReady Devicetree does not require most Runtime Services after `ExitBootServices()` has been called, and does not require Option ROM loadable driver support. The lack of Runtime Services means changes to firmware variables, like `BootXXXX`, must be done in the UEFI environment before the OS boots. The lack of Option ROM support means that booting from PCIe devices may not be supported if the firmware does not have native drivers for the device.
- SystemReady Devicetree uses the Devicetree system description instead of ACPI and SMBIOS. Devicetree is used by Embedded Linux products and many embedded SoCs do not currently have working ACPI descriptions. Linux supports both ACPI and Devicetree system descriptions, so SystemReady platforms can all be supported with a single kernel image if the appropriate configuration options are enabled.
- The core SystemReady band provides forward and backward compatibility with generic off-the-shelf OS images. SystemReady Devicetree only supports limited compatibility and has the dependency that the board support package (BSP) is upstreamed and downported to the distros.

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

For information on developing SystemReady Devicetree compliant firmware, refer to the SystemReady Devicetree Band Integration and Testing Guide. In particular, the Build firmware for Compulab IOT-GATE-IMX8 platform section provides an example of how to build a SystemReady compliant platform.

10.4 Why has Arm transitioned SystemReady IR certifications to SystemReady Devicetree band self-declared compliance?

Arm has transitioned SystemReady IR certification program to a SystemReady Devicetree band self-declared compliance because the SystemReady IR certifications have successfully fulfilled the SystemReady program's initial objectives.

Now it is time to move to a more sustainable approach, which is the SystemReady Devicetree band self-declared compliance.

The reasons for this move include the following:

1. Maturity of the ecosystem and market:
    - Key industry players including major Silicon Providers (SiPs), Cloud Service Providers (CSPs), Independent BIOS Vendors (IBVs), Original Design Manufacturers (ODMs), and Original Equipment Manufacturers (OEMs) are actively involved, resulting in a significant increase in SystemReady IR certified
2. Challenges of a mature ecosystem:
    - With the increase in requests, the certification process itself has become a bottleneck that slows down partners' marketing activities.
    - Duplication of efforts across different SKUs (Stock Keeping Unit) and new releases that are based on the same SoC.
    - Decoupling the market from the band name. Strictly focusing on technologies, not on markets they serve.
    - Confusion from having distinct SR, IR, ES and VE certification bands.

Because of these factors, Arm has transitioned the IR band into the SystemReady Devicetree band as well as consolidating the SR and ES bands into a single SystemReady compliance band, shifting from a one-time certification approach to a continuous, partner-driven compliance model. This new approach emphasizes ongoing integration and delivery, ensuring standards are maintained throughout a device's lifecycle, rather than at just one specific certification point. The compliance-based model ultimately enhances product support and reduces troubleshooting over the entire lifespan of a device.

10.5 When should I be running the compliance test suite?

SystemReady compliance is not a one-time milestone. Firmware should meet compliance with every release, so you are encouraged to integrate ACS into your regression test suites and release pipelines. Since version 3.0, ACS has been enhanced to minimize human interpretation of results and improve automation, making it straightforward to integrate into any CI/CD workflow.

10.6 Do partners need to work with Arm or a Test Lab for self-declared compliance?

SystemReady Devicetree Band compliance is self-declared by vendors. Partners do not need to work with Arm or an external test lab to claim compliance; instead, they are expected to run the ACS test suite themselves and base their declaration on the results. Even if a vendor chooses to work with a test lab, the compliance declaration still comes from the vendor. Test labs can, however, facilitate the process by running ACS, helping interpret results, and providing guidance and support, but the responsibility for declaring compliance always remains with the vendor.

10.7 Will self-declared compliance be published on the Arm web page?

No, self-declared compliance is not made publicly available by Arm.

10.8 What are waivers, which tests are waivable and what is the waiver process?

Waivers allow vendors to request exceptions for specific test results under defined conditions. As of version 3.0.1, waivers can be applied only to a limited set of test categories. Details on which test sub-suites are identified as either waivable or non-waivable can be found in the test_categoryDT.json file.

Waivers are permitted because these tests may relate to recommendations rather than strict requirements, or because certain platform configurations and settings are not fully supported by the ACS test framework. In such cases, the vendor may determine whether a test result represents a genuine failure or an acceptable deviation due to system configuration. Waiver process can be found here: https://github.com/ARM-software/arm-systemready/blob/main/SystemReady-devicetree-band/README.md#waiver-application-process.

10.9 What is BSA? Is it required for SystemReady Devicetree compliance?

BSA (Base System Architecture) defines the hardware requirements for Arm-based platforms.

For SystemReady Devicetree compliance, BSA is not mandatory. That said, BSA is strongly recommended. While some of its recommendations are not strictly needed to boot a Linux distribution, following them makes platforms more robust, eases debugging, and helps catch corner cases or unexpected scenarios that may not show up in production testing but are likely to appear once products are released.

10.10 What is BBSR? Is BBSR required for SystemReady Devicetree compliance?

The Base Boot Security Requirements (BBSR) specification describes the security requirements for a device to comply with industry-standard security interfaces and covers the following areas:

- UEFI authenticated variables
- UEFI secure boot
- UEFI capsule updates
- TPM 2.0 and measured boot

However, interface compliance does not provide assurance that the underlying platform is secure. When architecting a system, system-level threat modeling should be performed to evaluate threats, risks, and mitigations.

For example, in the embedded IoT market, the Platform Security Architecture (PSA) and the PSA Certified framework provides a comprehensive approach to platform security that is based on a defined set of security goals. PSA provides architecture and requirements specifications for building secure platforms. BBSR compliance complements PSA. PSA Certified shows the robustness of an implementation, through an assessment process that is performed by a security compliance laboratory.

Though complete BBSR compliance is not required for SystemReady Devicetree compliance, certain rules within the BBSR specification are required. These rules are outlined in the SystemReady Requirements Specification. Arm does recommend that partners who are looking to implement security features on their platforms such as UEFI Secure Boot, TPM 2.0 and Measured Boot should align to, and test for BBSR compliance.

10.11 What is PFDI? Is PFDI required for SystemReady Devicetree compliance?

PFDI stands for Platform Fault Detection Interface, and it does exactly what the name implies — it provides an interface that allows the operating system to query the platform for static fault status.

Compliance with PFDI is only required if it is implemented. If the ACS detects that PFDI is present, the corresponding tests will be executed, and compliance will then be mandatory.

10.12 How do I get access to the SystemReady Devicetree logo and trademarks?

Access to the SystemReady Devicetree logo and related trademarks is managed by Arm. Vendors who self-declare compliance must review and agree to Arm's trademark and logo usage guidelines. Once compliance is declared, partners may request authorization to use the official SystemReady branding.

You can submit your request via Arm's SystemReady Compliance Declaration contact form: Arm SystemReady Compliance — Contact Us.

11. SystemReady Devicetree band compliance testing - FAQ

This section provides answers to questions about Arm SystemReady Devicetree band compliance testing.

11.1 Where can I find the SystemReady Devicetree Architecture Compliance Suite (ACS)?

Each System Architecture Compliance Suite (ACS) is available in a dedicated repository, where the latest source code and releases can be accessed.

The SystemReady Devicetree prebuilt live OS images contain the latest builds of all relevant ACS. The latest prebuilt image can be found on the ACS Release Page.

11.2 Which version of the Architecture Compliance Suite (ACS) should I use for compliance testing?

The latest SystemReady Requirements Specification (SRS) will describe which version of the ACS is required for compliance testing. Arm would always recommend to use the latest available release.

11.3 How do I setup to run the SystemReady Devicetree live image?

Pre-built SystemReady Devicetree live ACS images are available on Github: https://github.com/ARM-software/arm-systemready.

These live images are deployed onto a storage device, such as a USB drive or SD card, which can then be used on the System under Test.

You must do the following before running the SystemReady Devicetree ACS Live Image:

- Make the bootable ACS live image available on the System under Test (SuT) on a bootable storage device.
- Prepare the SuT machine with up-to-date firmware and a host machine for SuT console access.
- For checking BBSR compliance (UEFI secure boot), make sure the system firmware is in Setup Mode. This is where the Secure Boot keys are cleared before starting the ACS. The mechanism to enroll Secure Boot keys is platform-specific and the procedure to enroll the keys must be available.

- Fill in the system details in the acs_tests/config/system_config.txt file on the flashed ACS partition.
- For systems that support in-band system firmware update, vendor-provided signed firmware update capsules, as well as “tampered” and “de-authenticated” capsules, created with capsule-tool.py must be available inside the flashed ACS partition as described in the Automated Capsule Update Testing Guide.
- Ensure copies of the OS Distro install logs and tests, described in the SystemReady Devicetree band Integration and Testing Guide's Distribution Testing section, are present in the flashed ACS images /os-logs/ directory.

The documentation for the band being tested provides further guidance for deployment:

- For the SystemReady Devicetree band follow the deployment steps in the SystemReady Devicetree Band Integration and Testing Guide

Further guidance according to compliance band is also available on GitHub: https://github.com/ARM-software/arm-systemready/tree/main/SystemReady-devicetree-band.

A typical deployment process consists of the following steps:

- Uncompress the prebuilt ACS image.
- On the Linux host machine, deploy the prebuilt ACS image to the storage device using one of the following sets of commands. The image name and path should reflect the image being deployed.

For SystemReady Devicetree band:

```
$ lsblk
$ sudo dd if=/path/to/systemready-dt_acs_live_image.wic of=/dev/sdX
$ sync
```

The `lsblk` command displays the storage devices in the system. Use the output of this command to identify the name of the target storage device.

The `dd` command copies the prebuilt ACS image to the storage device. Replace `/dev/sdX` with the name of the target storage device.

The `sync` command forces the copy operation to complete, so that you can unplug the target storage device.

11.4 What does the SystemReady Devicetree ACS live image testing process look like?

Once the ACS pre-built image has been flashed to a storage device and the Signed Capsules, populated system_config.txt and os-logs have been copied to the correct locations on the ACS partition, as described in How do I setup to run the SystemReady Devicetree live image?, the storage device should be connected to the System Under Test (SUT). Upon powering on the SUT

test suite should begin running automatically, resulting in the eventual boot of a Linux image with the following printed over UART:

```
init.sh[477]: ACS automated test suites run is completed.
```

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

To know which tests apply, always refer to the SystemReady Requirements Specification. In that document:

- Items marked as requirements are mandatory. The associated ACS tests must run and pass for compliance, unless a waiver is applied.
- Items marked as recommendations must also be executed, but passing them is not required. Failing a recommendation does not disqualify compliance, though vendors are encouraged to fix issues.
- Some requirements and recommendations are conditional: they only apply if a specific feature (like TPM) is implemented. If the feature isn't present, the tests are not applicable.

11.6 How long does it take to run the tests?

A typical ACS run can take anywhere between 4-8 hours depending on the speed of your storage medium.

11.7 Which OS Distributions can be used for compliance testing?

As mentioned in the SystemReady Requirements Specification:

SystemReady Devicetree band requires that installation and boot test logs from three of the actively supported versions of Linux or BSD are used to check for compliance. The recommended distributions are Fedora, Debian, Ubuntu, RHEL, Rocky Linux, SLES, openSUSE, OpenWrt, and Yocto.

When selecting Linux distributions or BSDs, maximize coverage by installing an OS from different groups in the following list. Avoid repetition within groups unless all four groups are covered:

- RHEL, Fedora, or Rocky Linux
- SLES or openSUSE
- Ubuntu or Debian

- OpenWrt or Yocto

11.7.1 Pre-built OS distributions

Pre-built distributions used in testing should be generic and actively supported versions from the OS vendor of choice, as specified in the SRS. OS versions that are nearing end-of-life should be upgraded to more recent versions. In certain cases, and only when deemed appropriate, daily builds or live distributions may be accepted as proof of fixes that have not yet been propagated to the distribution's official releases.

11.7.2 Custom-built OS distributions

Custom-built distributions, such as Yocto and OpenWrt, are allowed for SystemReady Devicetree band. However, generic pre-built images should be used for compliance testing. To maintain compliance and ensure that custom-built distributions remain functional, alterations to the default baseline layers should occur in the mainline. Modifying default layers of downstream custom builds is prohibited. Customization is allowed only in layers outside the generic baseline layer used for compliance, to ensure that interoperability is not compromised.

11.7.3 Can I use live or pre-installed images for self-compliance?

Using live or pre-installed images when testing for compliance with three OSes is not allowed, except for OpenWrt and Yocto.

After three OSes have been tested and compliance has been met, performing further testing with additional OSes using any type of image (including live or pre-installed images) is strongly encouraged.

11.8 How can partners decide whether a test failure is critical for self-compliance?

The SystemReady Devicetree Architecture Compliance Suite (ACS) is designed to provide a conclusive Compliant/Non-Compliance result for your platform after running. The ACS will use the results of all critical tests to infer the platform's compliance for this result. The ACS will also highlight failures or warnings in any non-critical tests. Partners are advised to take the time to understand and consider how that failures and warnings may impact their platform.

11.9 I have found an issue with the ACS, what should I do?

If an issue with the ACS is found, then:

- check github open issues https://github.com/ARM-software/arm-systemready/issues
- if the issue you encounter is already reported, you can comment and add details on your case
- otherwise, if the issue is new, then raise an Issue with detailed information for the ACS team analyze the root cause
- alternatively you can try to fix it your self and raise a PR to merge the fix, the maintenance team will review the PR and will accept or reject it.

11.10 Can I contribute into ACS development?

Yes, the ACS lives at https://github.com/ARM-software/arm-systemready/tree/main/SystemReady-devicetree-band and it is open to contributions from external developers, whether it is an enhancement to improve test coverage or a bug fix, all contributions to the ACS are welcome.

12. Related information

The following resources are related to material in this guide.

Specifications:

- Arm SystemReady Requirements Specification
- Arm Base System Architecture (BSA) specification
- Arm Server Base System Architecture (SBSA) specification
- Arm Base Boot Requirements (BBR) specification
- Arm SystemReady Requirements Specification
- Arm SystemReady Band Policy Guidelines
- Arm SystemReady Devicetree Band Compliance Policy Guidelines
- Arm Server Manageability Base Requirements

Repositories:

- Arm SystemReady ACS Repository
- SYSARCH-ACS Repository
- SystemReady GitLab Repository

User Guides:

- Arm SystemReady Band Integration and Testing Guide
- Arm SystemReady Devicetree Band Integration and Testing Guide
- SystemReady Pre-Silicon Reference Guide BSA integration and compliance

SystemReady Pages:

- Arm SystemReady Program
- Arm Community - SystemReady Forum
- System Architecture Compliance Suites (ACS)
- Arm SystemReady Band
- Arm SystemReady Devicetree Band
- Arm SystemReady Pre-Silicon

Appendix A Potentially waivable failures for SystemReady band compliance

This section describes potentially waivable failures for SystemReady band compliance.

A.1 BSA

Check either of the following files in your ACS media device to get the name of the failed test case. Then use the name to find out the triage action and reason for the waiver in the following table.

- `acs_results\uefi\BsaResults.log`
- `acs_results\acs_summary\html_detailed_summaries\BSA_detailed.html`

For PCIe tests (numbered 8xx), refer to the log to find the BDF number of the failed device. Then check the Linux `lspci` command log or the UEFI shell `pci` command log for the BDF number to get more information about the device. This information helps you answer the questions in the triage action field.

Failed test case name	Triage action	Reason for the waiver
6 : Check Cryptographic extensions	Check if the required cryptography extension support is permitted under trade regulations. If so, check if the SoC supports other cryptography extensions. If so, this could be a non-critical failure.	This is an expected failure due to the compliance with trade regulations in some countries, so this may not be considered a critical failure for compliance.
BSA 802 : Check Cmd Reg memory space enable	Check if the failed PCIe device is an invalid device. If so, check if the invalid device can be detected by OSes. If not, this could be a non-critical failure.	This issue may only cause problems with old OSes. If the target OS works properly with the issue, this may not be considered a critical failure for compliance. The failed PCI card can be replaced.
830: Check Cmd Reg memory space enable	Check if the failed PCIe root port is a valid device. If so, check if the root port and the devices attached to the root port can still be detected by the OS or work properly under the OS. If so, this could be a non-critical failure. Check if the failed PCIe device is a PCIe card. If so, this could be a non-critical failure.	This issue may only cause problems with old OSes. If the target OS can work properly with the issue, this may not be considered a critical failure for compliance. The failed PCI card can be replaced.
835 : Check Function level reset	Check if the failed PCIe devices are intended for VM assignment. If not, this may not be considered a critical failure for compliance. Check if the failed PCIe device is a PCIe card. If so, this could be a non-critical failure.	The test is based on a conditional requirement which only applies to VM assignment. The failed PCI card can be replaced.

Failed test case name	Triage action	Reason for the waiver
836 : Check ARI forwarding enable rule	Check if the failed PCIe devices can still be detected by the OS and work properly under OS, and there are no non-existing functions from this failed device getting detected by OS. If so, check if the OS can handle the non-existing functions properly. If so, this may not be considered a critical failure for compliance. Check if the failed PCIe device is a PCIe card. If so, this could be a non-critical failure.	This issue may only cause problems with old OSes. If the target OS can work properly with the issue, this may not be considered a critical failure for compliance. The failed PCI card can be replaced.
839 : Check MSI support for PCIe dev	Check if the failed PCIe devices can still be detected by the OS and work properly under OS. If so, check if the failed PCIe devices are intended for VM assignment. If not, this may not be considered a critical failure for compliance. Check if the failed PCIe device is a PCIe card. If so, this could be a non-critical failure.	This issue may only cause problems with old OSes and VMs. If the target OS and VM can work properly with the issue, this may not be considered a critical failure for compliance. The failed PCI card can be replaced.

A.2 SBSA

Check either of the following files in your ACS media device to get the name of the failed test case. Then use the name to find out the triage action and reason for the waiver in the following table:

- `acs_results\uefi\SbsaResults.log`
- `acs_results\acs_summary\html_detailed_summaries\SBSA_detailed.html`

Failed test case name	Triage action	Reason for the waiver
12 : Check for SHA3 and SHA512 support	Check if the required cryptography extension support is allowed by trade regulations. If so, check if the SoC supports other cryptography extension. If so, this may not be considered a critical failure for compliance	This is an expected failure due to the compliance with trade regulations in some countries, so this may not be considered a critical failure for compliance.

A.3 SCT

Check either of the following files in your ACS media device to get the failed test case name and the description of the failed protocol name. Then use the name to find out the triage action and reason for the waiver in the following table.

- `acs_results\edk2-test-parser\edk2-test-parser.log`, “Failure by group” section
- `acs_results\acs_summary\html_detailed_summaries\SCT_detailed.html`

Failed protocol • Failed test case name	Triage action	Reason for the waiver
HII_CONFIG_ACCESS_PROTOCOL • RouteConfigConformance • RouteConfigFunction • ExtractConfigFunction • ExtractConfigConformance HII_CONFIG_ROUTING_PROTOCOL • RouteConfig_Func	Remove PCIe add-in cards and check if the failure still exists. If not, this may not be considered a critical failure for compliance.	This issue only affects a PCIe add-in card's UEFI setup option. If the PCIe card works properly with default configuration (not requiring any change made in UEFI setup), it may not be a critical failure for compliance.
EFI_ADAPTER_INFORMATION_PROTOCOL • SetInformationFunction • SetInformationConformance • GetInformationFunction	Remove PCIe add-in cards and check if the failure still exists. If not, this may not be considered a critical failure for compliance.	This issue basically only affects the PCIe add-in card's inventory's display in UEFI. Therefore, if partners do not see an issue with OS boot and install, it may not be a critical failure for compliance.
COMPONENT_NAME2_PROTOCOL • GetControllerName_Conf	Remove PCIe add-in cards and check if the failure still exists. If not, this may not be considered a critical failure for compliance.	This issue only affects the device and its driver's name display in UEFI. Therefore, if partners don't see an issue with OS boot and install, it may not be a critical failure for compliance.
EFI_PXE_BASE_CODE_PROTOCOL • Arp_Conf • Start_Conf • Stop_Conf • Stop_Func EFI_SIMPLE_NETWORK_PROTOCOL • GetStatus_Conf • GetStatus_Func • MCastIpToMac_Conf • MCastIpToMac_Func • Receive_Conf • Shutdown_Func • Stop_Conf • Transmit_Conf	Check if the system can perform network boot, for example PXE boot or HTTP boot, with any network device on system. If so, this may not be considered a critical failure for compliance.	This is a firmware issue with a UEFI driver that may affect network boot. However, if the network device managed by the problematic UEFI driver is not used for any OS UEFI Boot Loader operation, such as PXE or HTTP boot, this may not be a critical failure for compliance.
EFI_SIMPLE_NETWORK_PROTOCOL • PlatformSpecificElements	Check if the platform needs the feature mentioned in the messages with the failure in the log, for example HTTP support, Boot from network protocols, and so on. If not, this may not be considered a critical failure for compliance. Note that platform must meet the minimum hardware functionality requirements in the SRS specification.	Some platforms do not support features such as network boot, PCIe, and USB, so the failures are expected due to the lack of the firmware support for these features.

A.4 FWTS

Check either of the following files in your ACS media device to get a failure description. Then use the description to find out the triage action and reason for the waiver in the following table.

- `acs_results\fwts\FWTSResults.log`
- `acs_results\acs_summary\html_detailed_summaries\fwts_detailed.html`

Failure description	Triage action	Reason for the waiver
acpi_sbbr: SBBR mandatory ACPI table “DBG2” not found.	Check if the system only has one UART. If so, check if there is UEFI setup option for UART to enable ACPI DBG2 table. If so, this is not an issue.	Expected failure as the ACPI DBG2 table is used for OS debugging. If the system only has one UART, it makes sense to assign the UART to the ACPI SPCR table instead of the DBG2 table by default for OS console use.
Dmicheck: ... has a default value ‘xxxxxx.’ and probably has not been updated by the BIOS vendor.	Check if the fields in SMBIOS types 1, 2, 3, 4, 45 can be filled by updating FRU or through other methods during the manufacturing process before shipping. If so, this is not an issue. Use a production system for compliance testing.	Expected failures as some data in SMBIOS types 1, 2, 3, 4, 45 are expected to be updated before shipping.