SystemReady Band Integration and Testing Guide

Version 5.0

Non-Confidential

Issue 02

102677_0500_02_en

SystemReady Band Integration and Testing Guide

Release information

Document history

Issue	Date	Confidentiality	Change
0500-02	25 June 2025	Non-Confidential	Minor update
0500-01	6 June 2025	Non-Confidential	Major update
0400-01	24 December 2024	Non-Confidential	Major update
0300-02	4 April 2024	Non-Confidential	Minor updates
0300-01	25 October 2023	Non-Confidential	Integrated SR ACS requirements into ACS chapter
0200-01	14 April 2023	Non-Confidential	Added SR requirements to create the SystemReady SR/ES Integration Guide
0100-01	1 January 2020	Non-Confidential	Initial release of SystemReady ES Integration Guide

Proprietary Notice

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.

This document includes language that can be offensive. We will replace this language in a future issue of this document.

To report offensive language in this document, email terms@arm.com.

Contents

1. Terms and abbreviations

This document uses the following terms and abbreviations.

UEFI
: Unified Extensible Firmware Interface

EDK2
: EFI Development Kit 2

ACPI
: Advanced Configuration and Power Interface

ASL
: ACPI Source Language

AML
: ACPI Machine Language

SMBIOS
: System Management BIOS

PXE
: Preboot Execution Environment

USAP
: USB Attached SCSI Protocol

ACS
: Architecture Compliance Suite

BSA
: Base System Architecture

SBSA
: Server Base System Architecture

BBR
: Base Boot Requirement

2. SystemReady Band Test and Integration Overview

This guide describes how to integrate SystemReady band compliant systems, how to develop and build the firmware, and how to run the SystemReady band compliance tests.

For more information about the SystemReady band compliance, please visit the Arm SystemReady Band website.

In this guide, you will learn:

- How to set up a Radxa Orion O6 for SystemReady band compliance tests and development
- How to set up an Arm Neoverse N2 reference design (RD-N2) FVP for SystemReady band compliance testing and development
- How to use the SystemReady band compliance test suites
- How to review the compliance test result and perform self-compliance
- How to install and run generic off-the-shelf OSes on Arm systems to further validate compliance

---

Note

This guide describes expectations for SystemReady band compliant systems. Additional requirements may apply depending on target market-segment. For example, SBSA requirements additionally apply for servers.

Before you begin

This guide assumes you are familiar with the following technologies and frameworks:

- UEFI
- EDK2 firmware development environment
- ACPI, ASL, and AML
- SMBIOS

This guide is aimed at the following audiences:

- IHVs and OEMs who develop SystemReady band compliant platforms
- UEFI developers who implement ACPI and SMBIOS support for SystemReady band compliant platforms
- Operating system developers who adapt their operating systems to run on SystemReady band compliant platforms

3. Set up the Radxa Orion O6

The Radxa Orion O6 is a SystemReady SR certified and SystemReady band compliant system. The Radxa Orion O6 is based on the CIX P1 SoC, an Arm Cortex-A720 based SoC. This section describes how to set up the Radxa Orion O6 for compliance testing and development.

To set up the Radxa Orion O6 system, you need the following hardware. For more information, see the Getting Started - Quick Start guide in the Radxa documentation center.

Power

A USB Type-C power adapter.

A 65W USB C PD (20V/3.25A) or USB C DC (20V/3.25A) power supply is recommended.

Storage

M.2 NVMe SSD

- Supported types: 2230, 2242, 2260, 2280
- Supported interfaces: PCIe Gen3 or Gen4. SATA M.2 SSDs are not supported.

PCIe NVMe card

USB Drive

- Supported: USB 2.0/3.0, USB-A, or USB-C drives. USB 3.0 is recommended for faster installation.

Graphic console

Monitor or TV: HDMI, DP, or USB-C video input

Cable: HDMI, DP, or USB-C cable

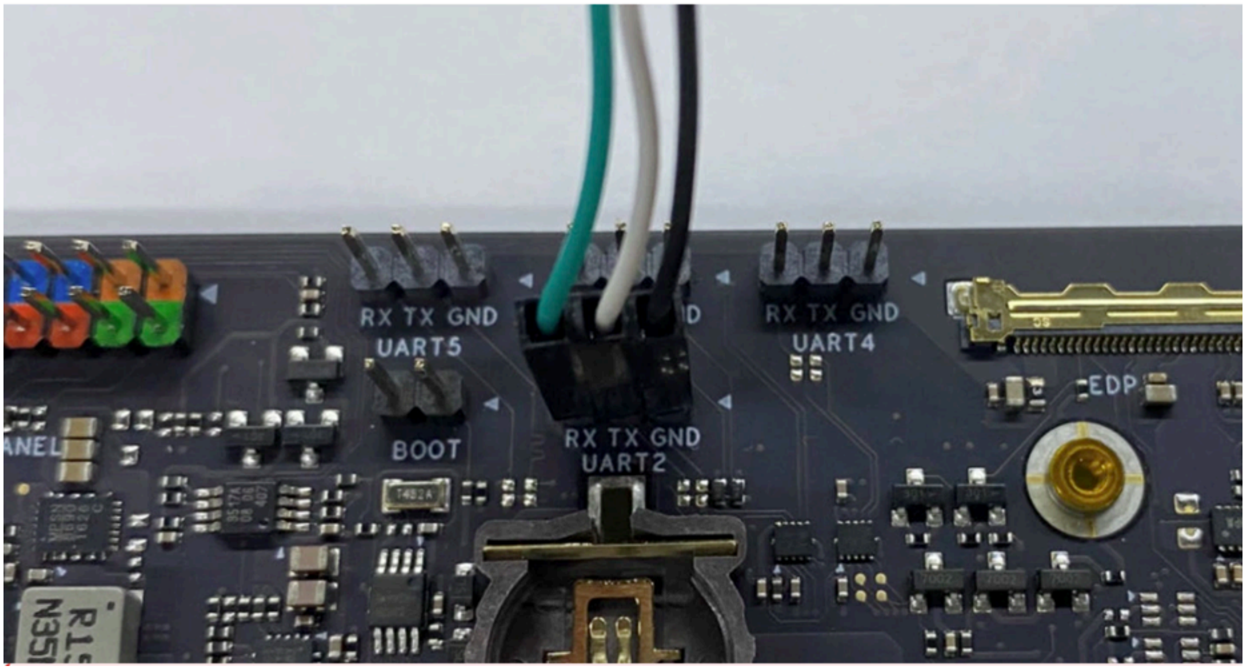
Keyboard and mouse

USB mouse and keyboard with generic drivers

Serial adapter

A USB to TTL serial adapter cable. You need to use three of the wires. Figure 3-1: Serial adapter connections on page 11 shows how to connect the three wires of your serial adapter cable to your Radxa Orion O6 system. For more information, see the UART Serial Console guide.

Figure 3-1: Serial adapter connections



The following table shows the connection details:

Description	RX	TX	GND
Color	Green	White	Black

Finally, connect the serial cable USB connector to your PC or laptop.

3.1 Set up the terminal

If you are using Windows, you need a terminal emulator such as PuTTY.

The following table shows the configuration required, and the following text describes how to set up your connection with PuTTY:

Variable	Value
Baud rate	115200
Data bits	8
Parity	None
Stop bits	1

To set up your connection with PuTTY:

1. On the **Session** configuration panel in PuTTY, select **Serial** from the **Connection type** options.

2. Use the **Serial line** and **Speed** options to specify which serial line to use and the Baud rate to use to transfer data.
3. For more information on serial connection with PuTTY, see Connecting to a local serial line.

If you are using Linux or a Mac, use terminal emulators such as minicom or screen to connect to the TTL serial connection. If there are no serial devices connected to your computer, your serial connector is `/dev/ttyUSB0`. If you have more than one serial device, use a tool such as `dmseg` to check `ttyUSB<num>`.

To connect using `screen`, enter the following command:

```
$ screen /dev/ttyUSB0 115200
```

To connect using minicom, enter the following command:

```
$ minicom -D /dev/ttyUSB0
```

For more information and troubleshooting, see UART Serial Console guide.

3.2 Update the system firmware

If you would like to update the firmware (v9.0.0) used for certification, you can download the firmware update package (pre-built images and tools) from https://dl.radxa.com/orion/o6/images/bios/SystemReady/240411/orion-o6-bios.7z and extract the .7z file and copy the files into the top directory of the FAT32 partition on the USB flash drive.

If you would like to use the latest firmware, you can download the firmware update package (`edk2-cix_<firmware version>_all.deb`) from https://github.com/radxa-pkg/edk2-cix/releases, extract the deb file, and copy all files in the `edk2-cix_<firmware version>_all\data\usr\share\edk2\radxa\orion-o6` folder to the top directory of the FAT32 partition on the USB flash drive. For details, please refer to https://radxa-pkg.github.io/edk2-cix/install.html.

Insert the USB drive into the system and power it on. During the boot process, press ESCAPE key to get to Boot Manager menu and select UEFI shell to boot. Then, go to the folder on the USB flash drive where the BIOS files and tools are located, and then run `setup.nsh` or `startup.nsh` to update the system firmware. For details, see the Install BIOS guide in the Radxa documentation center.

3.3 System firmware source code

Radxa Orion O6 firmware is open source, which makes it developer-friendly for building from source, doing firmware development, and addressing compliance issues. For details, please refer to https://github.com/radxa-pkg/edk2-cix.

- The firmware source code is located at https://github.com/radxa-pkg/edk2-cix/tree/main/src.
- The build instructions are available at https://github.com/radxa-pkg/edk2-cix/blob/main/README.md and Radxa's documentation site. However, if your build environment does not have Visual Studio Code and Docker Desktop or you encounter issues with instructions on the Radxa website, you can instead follow the steps below in Ubuntu OS. If your build environment already has docker, nodejs, npm, and devcontainer CLI installed, only steps 3 and 4 are required.

    1. Install docker:
        - https://docs.docker.com/engine/install/
    2. Install devcontainer CLI:

    ```
    $ sudo apt remove -y nodejs npm
    $ sudo apt autoremove -y
    $ curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
    $ sudo apt install -y nodejs
    $ node -v
    $ npm -v    # Should be v10.x
    $ sudo npm install -g n
    $ sudo n lts
    $ sudo apt install npm
    $ sudo npm install -g @devcontainers/cli
    ```

    3. Use the devcontainer CLI to open an existing folder in a container:

    ```
    $ sudo devcontainer up --workspace-folder <Path to edk2-cix folder>
    $ sudo devcontainer exec --workspace-folder <Path to edk2-cix folder> /bin/
    bash
    ```

    4. Run `make` in container `vscode > /workspaces/edk2-cix (main) $`

    ```
    $ make deb
    ```

To update the self-built firmware image, do the following:

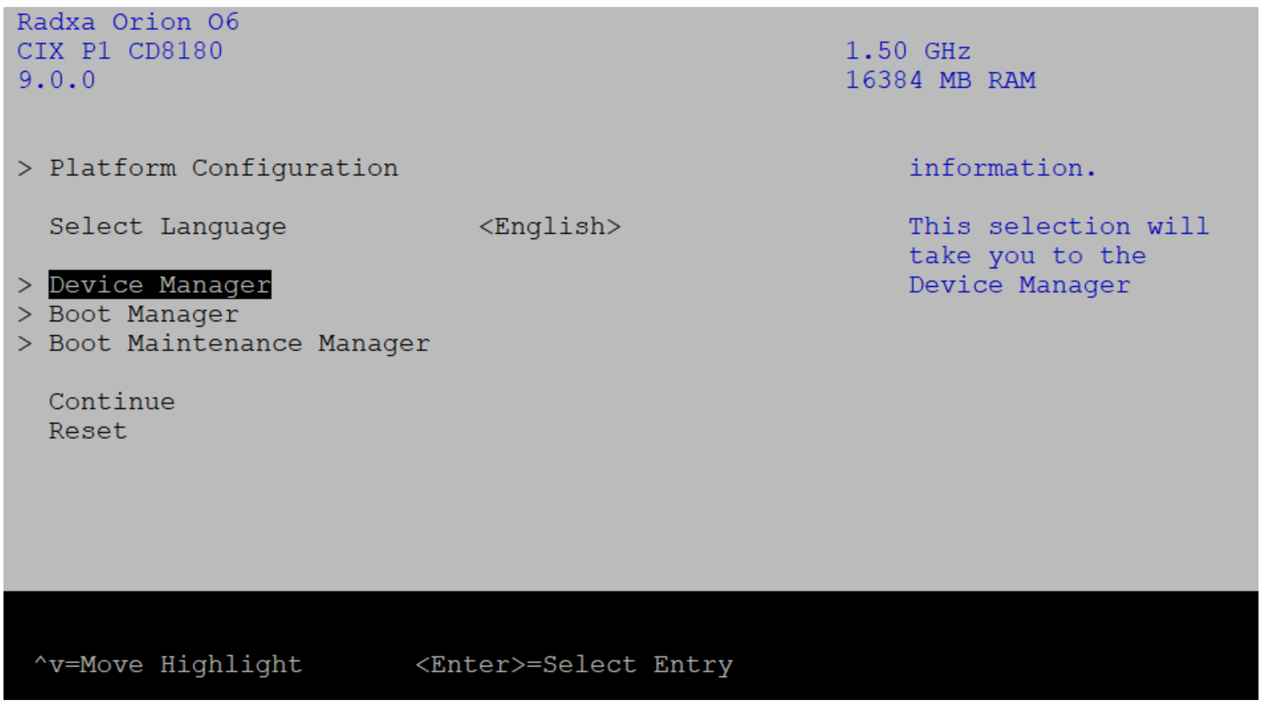
1. After the build process is complete, the self-built firmware image (cix_flash_all.bin) is located in `/edk2-cix/Src/Build/O6/RELEASE_GCC5`
2. Download the firmware update package (7z/zip/deb file) from https://dl.radxa.com/orion/o6/images/bios/ or https://github.com/radxa-pkg/edk2-cix/releases to get the firmware update tools
3. Replace the `cix_flash_all.bin` file in the firmware update package with the self-build image, and follow the instructions in https://docs.radxa.com/en/orion/o6/bios/install-bios to update the firmware on the system.

Note that for general use cases like running compliance testing, it is recommended to use the pre-build image mentioned in previous section. Using a self-built firmware is risky. It may cause the system to freeze, requiring a USB SPI flash programmer such as CH341A to recover the system. For details about using a flash programmer, see Method 3: Offline BIOS Update and the related community discussion thread.

3.4 Configure UEFI and troubleshooting UEFI using UEFI shell

To boot the UEFI setup menu, press **Esc** during the boot process. The following UEFI menu is displayed:

Figure 3-2: UEFI setup menu



In this menu, you can change device settings using Device Manager and manually boot the device using Boot Manager.

Figure 3-3: Boot Manager menu

```
|                              Boot Manager                                    |
\------------------------------------------------------------------------------/


                                                           Device Path :
  Boot Manager Menu                                        Fv(9A15AA37-D555-4A4E-
                                                           B541-86391FF68164)/FvF
  UEFI Lexar SSD NM7A1 512GB NL6306R000308P2209 1          ile(7C04A583-9E3E-4F1C
  UEFI PXEv4 (MAC:004854200C68)                            -AD65-E05268D0B4D1)
  UEFI PXEv6 (MAC:004854200C68)
  UEFI HTTPv4 (MAC:004854200C68)
  UEFI HTTPv6 (MAC:004854200C68)
  UEFI PXEv4 (MAC:004854200C67)
  UEFI PXEv6 (MAC:004854200C67)
  UEFI HTTPv4 (MAC:004854200C67)
  UEFI HTTPv6 (MAC:004854200C67)
  UEFI Shell
  UEFI Kingston DataTraveler 3.0
  E0D55EA574B315B1888200B6
                                                        v
|
|
\ ^v=Move Highlight       <Enter>=Select Entry      Esc=Exit                     /
```

Figure 3-4: UEFI shell screen on page 15 shows how to boot into the UEFI shell by selecting the UEFI shell boot option from the Boot Manager menu:

Figure 3-4: UEFI shell screen

```
UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00000000)
Mapping table
      FS0: Alias(s):HD0b:;BLK1:
          PciRoot(0x1)/Pci(0x0,0x0)/Pci(0x0,0x0)/NVMe(0x1,34-01-00-30-03-5B-F2-C
A)/HD(1,GPT,77440DD2-3DFC-44D2-B42D-3BF503CE7968,0x800,0x32000)
      FS1: Alias(s):HD1b0b:;BLK6:
          VenHw(0D51905B-B77E-452A-A2C0-ECA0CC8D514A,00801D090000000000)/USB(0x1
,0x0)/HD(1,GPT,48D9408B-D94A-4733-93DB-CDB2CFF2FF0D,0x800,0x13FFFF)
     BLK0: Alias(s):
          PciRoot(0x1)/Pci(0x0,0x0)/Pci(0x0,0x0)/NVMe(0x1,34-01-00-30-03-5B-F2-C
A)
     BLK2: Alias(s):
          PciRoot(0x1)/Pci(0x0,0x0)/Pci(0x0,0x0)/NVMe(0x1,34-01-00-30-03-5B-F2-C
A)/HD(2,GPT,C06B6B23-6292-4B8A-8977-EBAF71739589,0x32800,0x8000)
     BLK3: Alias(s):
          PciRoot(0x1)/Pci(0x0,0x0)/Pci(0x0,0x0)/NVMe(0x1,34-01-00-30-03-5B-F2-C
A)/HD(3,GPT,6CE95F6D-EA84-4488-B44B-CE5F49909B5A,0x3A800,0x3B874000)
     BLK4: Alias(s):
          PciRoot(0x1)/Pci(0x0,0x0)/Pci(0x0,0x0)/NVMe(0x1,34-01-00-30-03-5B-F2-C
A)/HD(4,GPT,82407436-24DF-45BA-A113-075CE35D120E,0x3B8AE800,0x132000)
     BLK5: Alias(s):
          VenHw(0D51905B-B77E-452A-A2C0-ECA0CC8D514A,00801D090000000000)/USB(0x1
,0x0)
Press ESC in 5 seconds to skip startup.nsh or any other key to continue.
Shell>
```

The following table shows UEFI Shell commands which are helpful for debugging:

Command	Description
`pci`	Show PCIe devices or PCIe function configuration space information
`drivers`	Show a list of UEFI drivers
`devices`	Show a list of devices managed by UEFI drivers
`dh -d -v > dh_d_v.txt`	Save a dump of all UEFI Driver Model-related handles to dh_d_v.txt
`memmap`	Save the memory map to memmap.txt
`smbiosview`	Show SMBIOS information
`acpiview -l`	Show a list of ACPI tables
`acpiview -r 2`	Validate that all ACPI tables required by SBBR 1.2 are installed.
`acpiview -s DSDT -d`	Generate a binary file of DSDT ACPI table.
`dmpstore -all > dmpstore.txt`	Dump all UEFI variables to dmpstore.txt

See the UEFI Shell Specification for more details. The Shell commands section provides a list of shell commands, descriptions, and examples.

3.5 Set UEFI variables

You can view and change the Radxa Orion O6 UEFI configuration settings using the UI configuration menu and UEFI shell. To configure the system using the UEFI Shell, use `setvar` to read and write the UEFI variables for the GUID CD7CC258-31DB-22E6-9F22-63B0B8EED6B5.

To read a setting, use the following command:

```
setvar <NAME> -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5
```

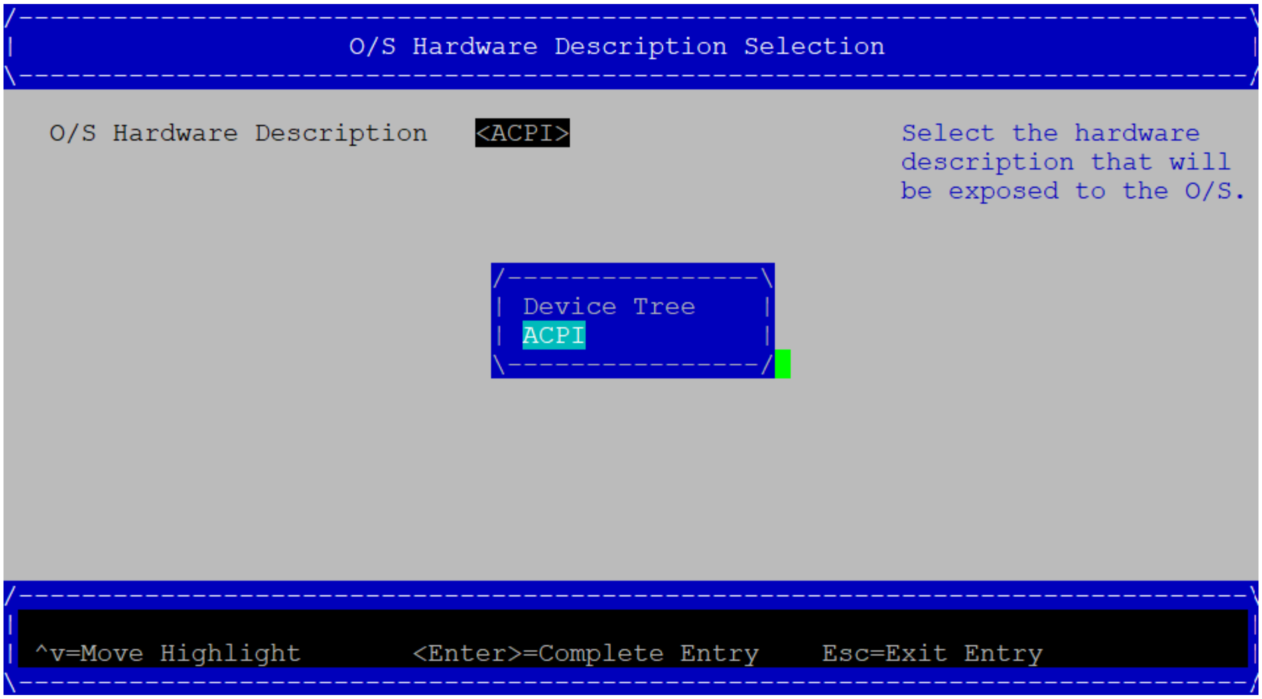
To write a setting, use the following command:

```
setvar <NAME> -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5 -bs -rt -nv =<VALUE>
```

3.6 Set the OS Hardware Description method

In the **Device Manager > O/S Hardware Description** selection menu, ensure that **ACPI** is selected as shown in the figure below:

Figure 3-5: OS hardware description



3.7 Set the console preference

Linux uses the `/chosen/stdout-path` Device Tree property or the ACPI SPCR table to designate serial port as the primary console, even if a graphical console is available. Therefore, for some Linux OSes that cannot output message to all console devices, users may need to enable or disable ACPI SPCR table to switch the primary console between serial console and graphical console.

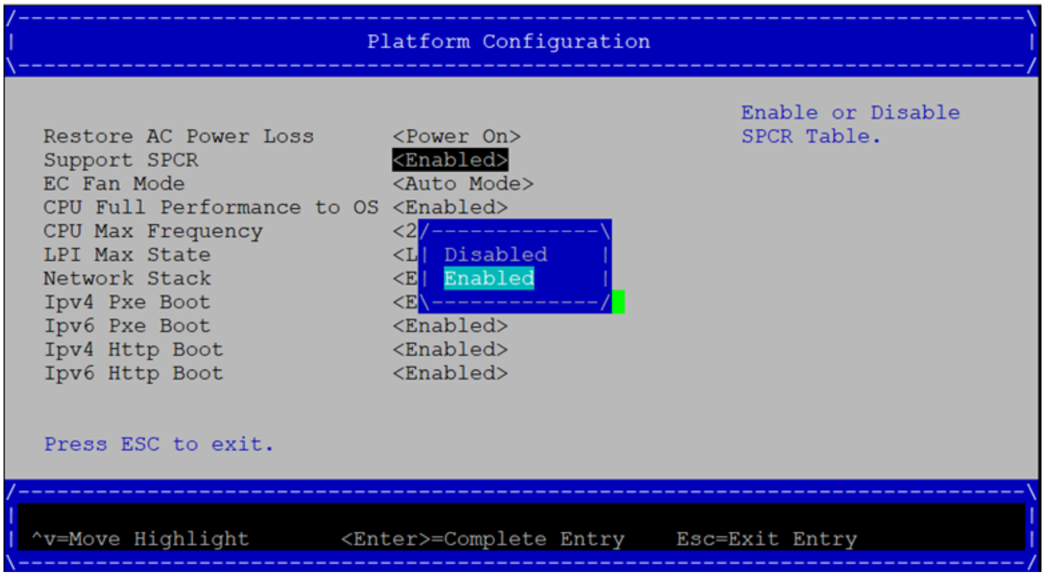
Note

The ACPI SPCR table must be **Enabled** when running the ACS test suite because the graphic console may not work properly with the generic graphic drivers in ACS Linux. Also, the ACPI SPCR table is mandatory for SystemReady band compliance.

For Radxa Orion O6, ensure that the setting of the **Support SPCR** option is **Enabled**:

1. Open **Device Manager** in the UEFI menu.
2. Select **Support SPCR** as shown in the following figure:

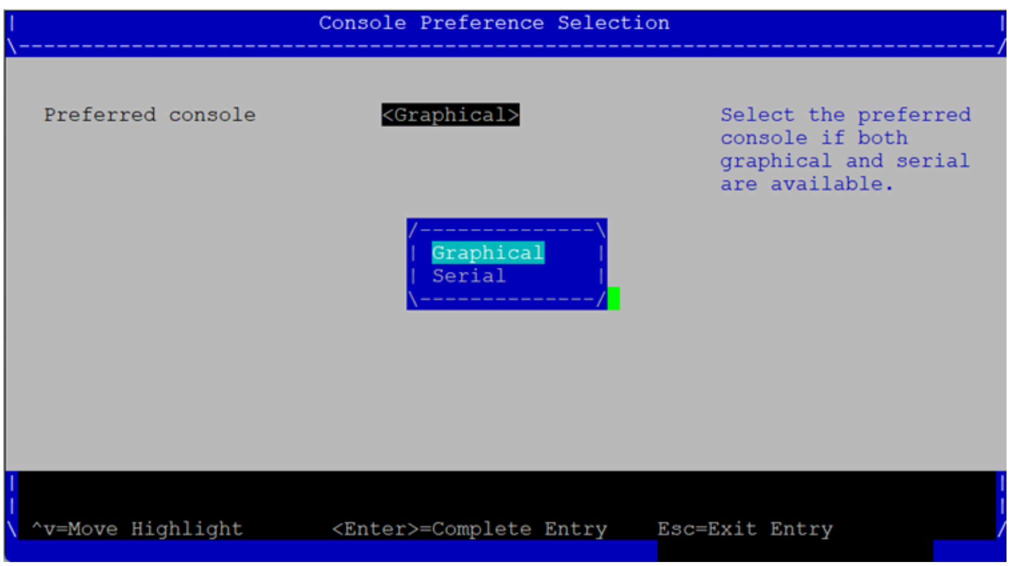
Figure 3-6: SPCR support option



For systems that are using edk2 open source, the setup option is called **Console Preference Selection**.

1. Open **Device Manager** in the UEFI menu.
2. Select **Console Preference Selection** as shown in the following figure. **Graphical** means to disable SPCR. **Serial** means to enable SPCR.

Figure 3-7: Console preference selection



4. Set up the RD-N2 FVP

This section describes how to set up the RD-N2 FVP, which supports the emulation of a Arm Neoverse N2 based system. If you do not have access to a physical system, this FVP provides a useful alternative.

4.1 Set up the host machine and download the software stack

A host machine with Ubuntu 22.04 with 64 GB of free disk space and 32 GB of RAM is the minimum requirement to sync and build the platform software stack. However, we recommend 48 GB of RAM.

Follow the instructions in the getting-started.rst in the infra-refdesign-docs GitLab to install the necessary tools and download the source code for the software stack. For more information, see the Get started with the Neoverse Reference Design software stack learning path.

If the host machine's memory is less than 32 GB, follow the instructions for using the swap file to enable virtual memory.

You need a display manager to run the FVP. Using a text console to connect to the host machine does not work. For remote access to the host machine, you need a console application that supports display export. For example, you can follow these instructions: https://itsfoss.com/install-gui-ubuntu-server/ to install the `lightdm` display manager. Then install a remote desktop tool such as `xrdp`. An alternative is to use MobaXterm.

4.2 Download the RD-N2 FVP

The RD-N2 FVP installer is available from the Neoverse Infrastructure FVPs section on the Fixed Virtual Platforms site.

Run the following commands to download and install RD-N2 FVP:

```
$ wget https://developer.arm.com/-/media/Arm%20Developer%20Community/Downloads/OSS/
FVP/Neoverse-N2/Neoverse-N2-11-20-18-release/FVP_RD_N2_11.20_18_Linux64.tgz

$ tar -xvzf FVP_RD_N2_11.20_18_Linux64.tgz

$ ls
FVP_RD_N2_11.20_18_Linux64.tgz  FVP_RD_N2.sh  license_terms

$ ./FVP_RD_N2.sh

/FVP_RD_N2$ ls
bin  fmtplib          Iris           models   scripts
doc  install_history  license_terms  plugins  sw
```

For more information, see the Neoverse Reference Design Platform Software.

4.3 Build the software stack and run the FVP

Follow the instructions in the links below to build and run the FVP:

- ACS compliance test on Neoverse RD platforms
- WinPE boot on Neoverse RD platforms
- Install and boot an OS on Neoverse RD platforms

Note

You must run the FVP with the root account to access the console logs.

---

5. ACS

SystemReady band uses a collection of Architecture Compliance Suite (ACS), to help validate system compliance.

5.1 ACS overview

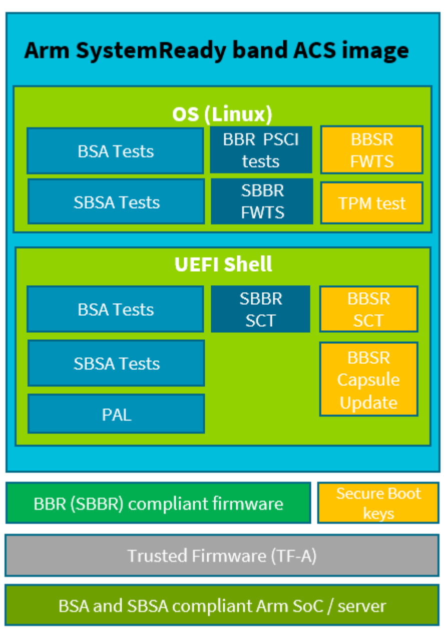
The Arm SystemReady band ACS live image provides a collection of tests designed to ensure architectural compliance across various implementations and variants of the architecture. The product is delivered as source code with a build environment. It creates a bootable live OS image encompassing a series of System Architecture Compliance Suites (ACS), including the BSA-ACS, SBSA-ACS, and BBR-ACS. These suites assess compliance against the BSA, SBSA, and BBR specifications for SystemReady band compliance. Arm recommends verifying architectural implementations against the ACS to demonstrate compliance with these specifications.

The Architecture Compliance Suites for SystemReady band compliance are facilitated through a live OS image, empowering basic automation for executing required and recommended tests. The ACS image contains the following components for checking the requirements during boot time and runtime.

- UEFI applications that operate on a UEFI shell.
- Linux kernel that incorporates kernel modules and Firmware Test Suite (FWTS).

Figure 5-1: ACS components on page 21 is a diagram illustrating the various components of the Arm SystemReady band ACS live image:

Figure 5-1: ACS components



5.2 BSA-ACS and SBSA-ACS

The BSA-ACS checks for compliance against the Arm Base System Architecture (BSA) specification, and the SBSA-ACS checks for compliance against the Server Base System Architecture (SBSA) supplement specification. BSA-ACS compliance is required by the SystemReady band. SBSA-ACS is only required for SystemReady band compliant servers. The tests are delivered through two parts:

- Tests on UEFI Shell. These tests consist of the UEFI shell command-line application `bsa.efi` and `sbsa.efi`. These tests are written on top of Validation Adaption Layers (VAL) and Platform Adaptation Layers (PAL). The abstraction layers provide the tests with platform information and runtime environment to enable execution of the tests. In Arm deliveries, the VAL and PAL are written on top of UEFI.
- Tests on the Linux command line. These tests consist of the Linux command-line application bsa and sbsa, and the kernel module `bsa_acs.ko` and `sbsa_acs.ko`.

5.3 BBR-ACS

The BBR-ACS checks compliance against the Arm Base Boot Requirements (BBR) specification. For SystemReady band compliance, firmware is tested against the SBBR recipe of BBR.

These tests are delivered through two bodies of code:

- SBBR tests contained in UEFI Self Certification Tests (SCT) tests. UEFI implementation requirements which are tested by SCT.
- SBBR based on FWTS. The Firmware Test Suite (FWTS) is a package hosted by Canonical that provides tests for ACPI and UEFI. The FWTS tests are customized to run only UEFI tests.

5.4 ACS prerequisites

The prerequisites to run the ACS live image are as follows:

- Prepare a storage device with a minimum of 1GB of storage. This storage device is used to boot and run the ACS and to store the execution results.

> **Note**
>
> We recommend you use NVMe/SATA drive or USB disk enclosure with a fast SSD drive. Using a USB stick may take longer than 6 hours for testing.

- Prepare the SUT (System Under Test) machine with the latest firmware loaded, a host machine for console access, then collect the results

5.5 Set up the test environment

To set up the storage device, use the following procedure:

1. Download the prebuilt SystemReady band live image to a local directory on Linux.

   The pre-built ACS image for SystemReady band compliance is available on GitHub at the following location:

   ```
   https://github.com/ARM-software/arm-systemready/tree/main/SystemReady-band/
   prebuilt_images/<release tag>/systemready_acs_live_image.img.xz
   ```

   For information on the latest release and release tags, see the SystemReady band README
2. Decompress and deploy the image to the storage device.

   Use a utility such as `xz` on Linux or 7-Zip on Windows to uncompress the `systemready_acs_live_image.img.xz` file.
3. On the Linux host machine, write the SystemReady band ACS live image to the storage device using the following commands:

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

   In this code, replace `/dev/sdX` with the name of your storage device. Use the `lsblk` command to display the storage device name.

   On the Windows host machine, you can use Rufus to write the image to the storage device.
4. To test the server system, change the value of `config_enabled_for_automation_run` and `automation_sbsa_run` from false to true in `acs_tests/config/acs_run_config.ini` to run the SBSA test. For details, see the SystemReady Band Execution Enviroment and Configuration User Guide.

Figure 5-2: Changes in acs_run_config.ini for server system

acs_run_config.ini

File Edit View

```
[AUTOMATION]
# This variable determines whether the automation run utilizes this configuration file.
config_enabled_for_automation_run = true

[SCT]
# This variable Enable/Disable SCT run(Valid values true or false).
automation_sct_run = true
# Valid value is Sequence file.
sct_sequence_file = SBBR.seq

[SCRT]
# This variable Enable/Disable SCRT run(Valid values true or false).
automation_scrt_run = true

[BSA]
# This variable Enable/Disable BSA run(Valid values true or false).
automation_bsa_run = true
# Valid values 0, 100, 200, 300, ........, 900.
bsa_modules =
# Add tests to run here.
bsa_tests =
# Add tests you want to skip here.
bsa_skip = 900
# Default value
bsa_verbose = 3

[SBSA]
# This variable Enable/Disable SBSA run(Valid values true or false).
automation_sbsa_run = true
# Valid values 0, 100, 200, 300 .........., 1300
sbsa_modules =
# Valid values are 3,4,5,6,7,fr.
sbsa_level = 4
# Add tests to run here.
sbsa_tests =
# Add tests you want to skip here.
sbsa_skip = 900
# valid values 1,2,3,4,5
sbsa_verbose = 3
```

5.6 Run the tests

To execute the SystemReady band prebuilt ACS live image, do the following:

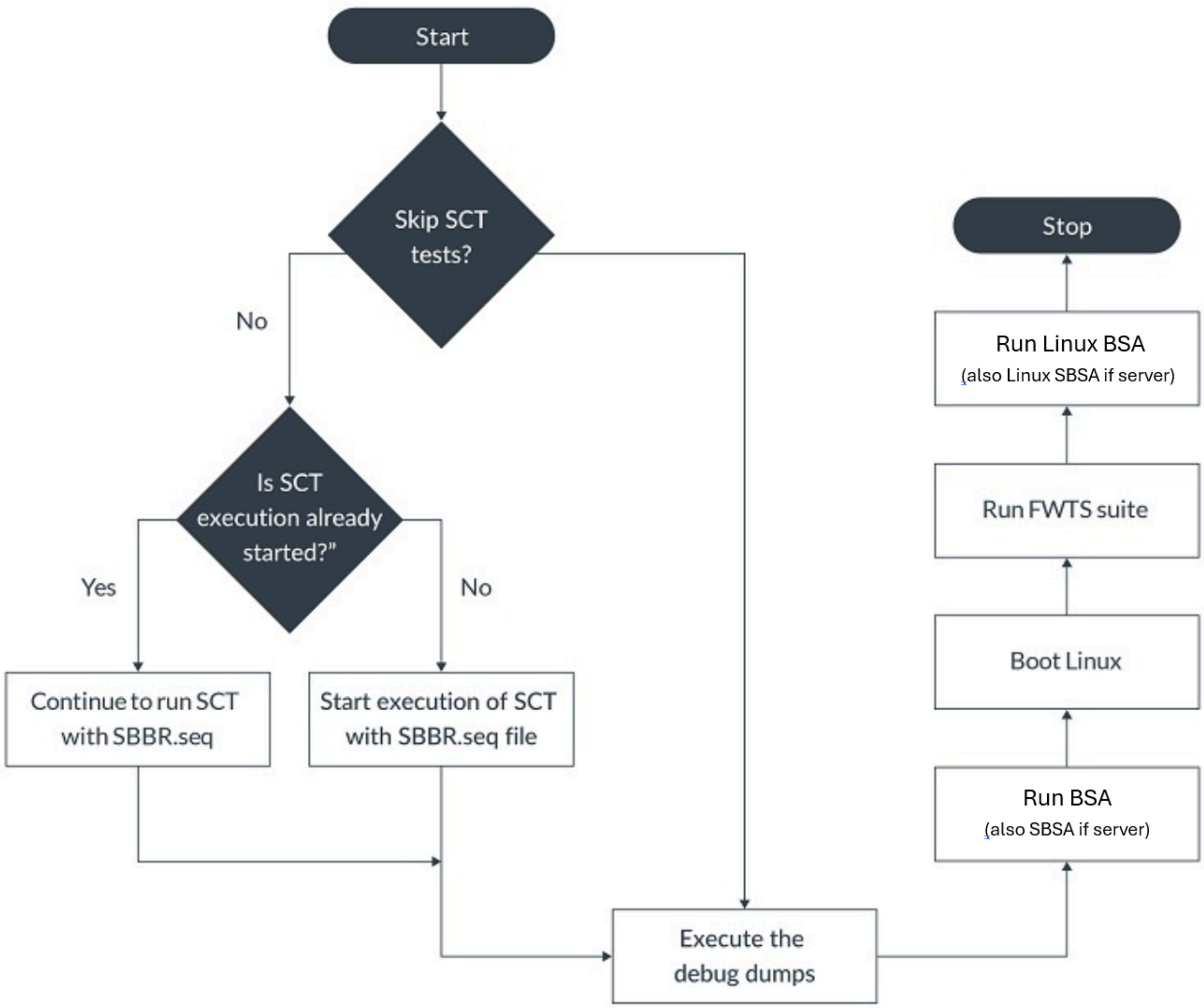
1. Insert the SystemReady ACS drive into the system.
2. Boot to the firmware setup menu.
3. Move the boot option of the ACS drive to the top of the boot order and then save the setting.

4. Reset the system.

   The live image boots and runs automatically.

Figure 5-3: Test execution process on page 25 shows the complete process of ACS execution through the SystemReady band ACS live image:

Figure 5-3: Test execution process



Note

To skip the debug and test steps shown in the diagram, press any key within five seconds.

As shown in the flowchart, there are two main modes of execution:

- Fully automated mode

- Normal mode

The following sections describe these modes in more detail.

5.7 Run tests in automated mode

If no option in GRUB is chosen and no tests are skipped, tests are run in fully automated mode.

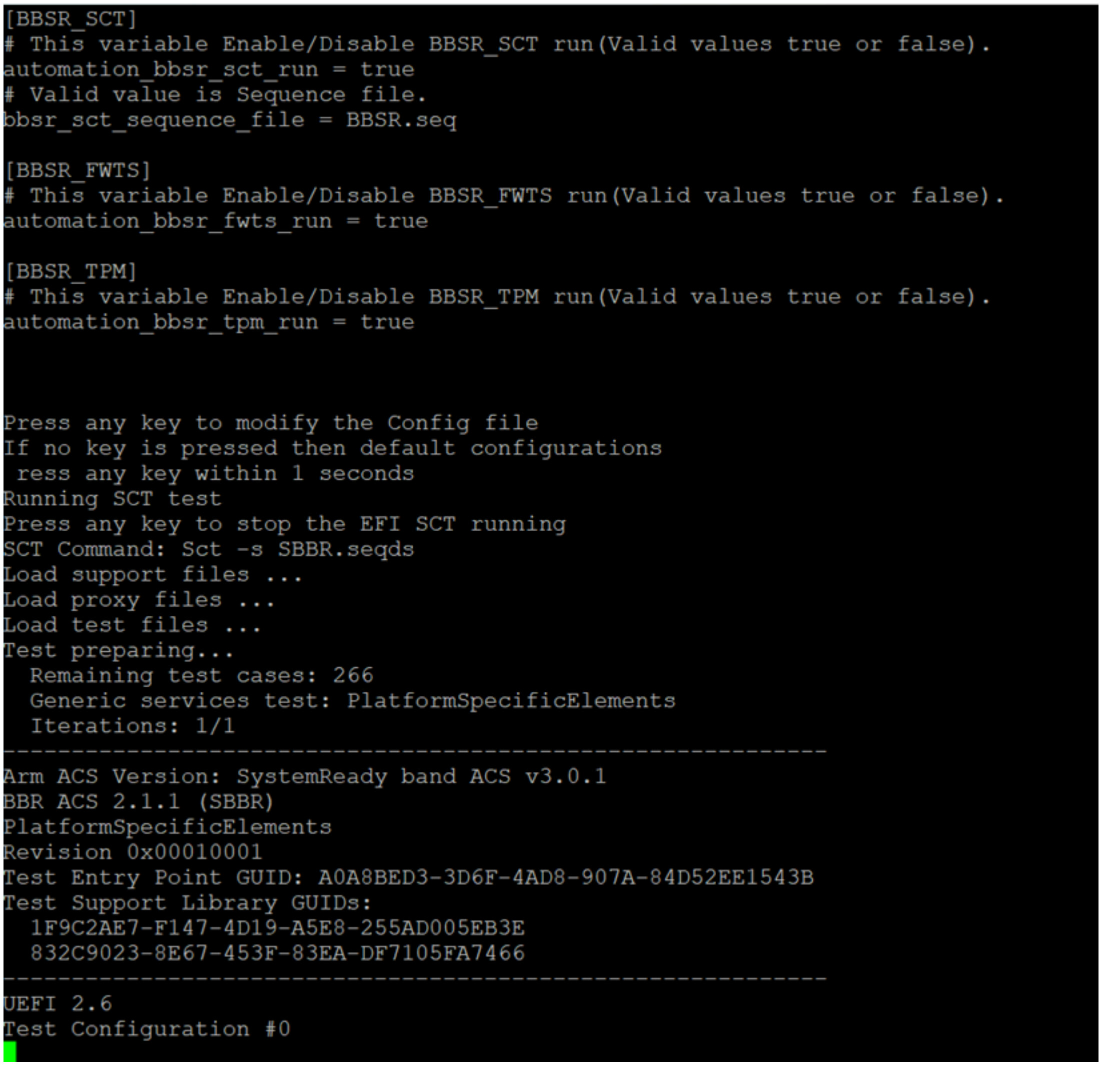
Figure 5-4: GRUB bootloader options on page 26 shows the GRUB bootloader options screen:

Figure 5-4: GRUB bootloader options

After a few seconds, the image executes the ACS process in the following order:

1. SCT tests:

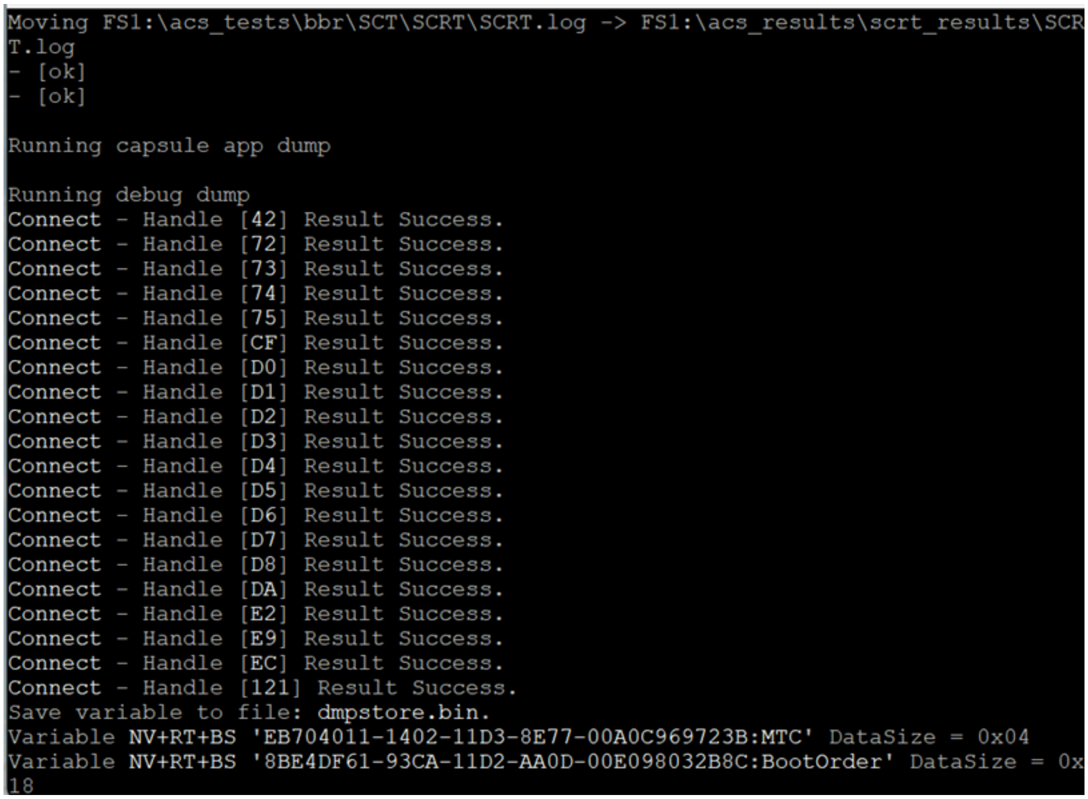
Figure 5-5: SCT tests



```
[BBSR_SCT]
# This variable Enable/Disable BBSR_SCT run(Valid values true or false).
automation_bbsr_sct_run = true
# Valid value is Sequence file.
bbsr_sct_sequence_file = BBSR.seq

[BBSR_FWTS]
# This variable Enable/Disable BBSR_FWTS run(Valid values true or false).
automation_bbsr_fwts_run = true

[BBSR_TPM]
# This variable Enable/Disable BBSR_TPM run(Valid values true or false).
automation_bbsr_tpm_run = true



Press any key to modify the Config file
If no key is pressed then default configurations
 ress any key within 1 seconds
Running SCT test
Press any key to stop the EFI SCT running
SCT Command: Sct -s SBBR.seqds
Load support files ...
Load proxy files ...
Load test files ...
Test preparing...
  Remaining test cases: 266
  Generic services test: PlatformSpecificElements
  Iterations: 1/1
------------------------------------------------------------
Arm ACS Version: SystemReady band ACS v3.0.1
BBR ACS 2.1.1 (SBBR)
PlatformSpecificElements
Revision 0x00010001
Test Entry Point GUID: A0A8BED3-3D6F-4AD8-907A-84D52EE1543B
Test Support Library GUIDs:
  1F9C2AE7-F147-4D19-A5E8-255AD005EB3E
  832C9023-8E67-453F-83EA-DF7105FA7466
------------------------------------------------------------
UEFI 2.6
Test Configuration #0
```

2. UEFI debug dumps:

Figure 5-6: UEFI debug dumps



```
Moving FS1:\acs_tests\bbr\SCT\SCRT\SCRT.log -> FS1:\acs_results\scrt_results\SCR
T.log
- [ok]
- [ok]

Running capsule app dump

Running debug dump
Connect - Handle [42] Result Success.
Connect - Handle [72] Result Success.
Connect - Handle [73] Result Success.
Connect - Handle [74] Result Success.
Connect - Handle [75] Result Success.
Connect - Handle [CF] Result Success.
Connect - Handle [D0] Result Success.
Connect - Handle [D1] Result Success.
Connect - Handle [D2] Result Success.
Connect - Handle [D3] Result Success.
Connect - Handle [D4] Result Success.
Connect - Handle [D5] Result Success.
Connect - Handle [D6] Result Success.
Connect - Handle [D7] Result Success.
Connect - Handle [D8] Result Success.
Connect - Handle [DA] Result Success.
Connect - Handle [E2] Result Success.
Connect - Handle [E9] Result Success.
Connect - Handle [EC] Result Success.
Connect - Handle [121] Result Success.
Save variable to file: dmpstore.bin.
Variable NV+RT+BS 'EB704011-1402-11D3-8E77-00A0C969723B:MTC' DataSize = 0x04
Variable NV+RT+BS '8BE4DF61-93CA-11D2-AA0D-00E098032B8C:BootOrder' DataSize = 0x
18
```

3. BSA and SBSA tests:

Figure 5-7: BSA tests

```
Running BSA test
Press any key to start BSA in verbose mode.
If no key is pressed then BSA will be run in normal mode
BSA Command: Bsa.efi -skip 900


 BSA Architecture Compliance Suite
          Version 1.1.0

 Starting tests for level  1 (Print level is  3)


 Creating Platform Information Tables
 PE_INFO: Number of PE detected       :    8
 Primary PE: MIDR_EL1                 :    0x410FD811
 GIC INFO: GIC version                :    v4
 GIC_INFO: Number of GICD             :    1
 GIC_INFO: Number of GICR RD          :    1
 GIC_INFO: Number of ITS              :    1
 TIMER_INFO: System Counter frequency :    1000 MHz
 TIMER_INFO: Number of system timers  :    0
 WATCHDOG_INFO: Number of Watchdogs   :    1
 PCIE_INFO: Number of ECAM regions    :    4
 PCIE_INFO: Number of BDFs found      :    8
 PCIE_INFO: Number of RCiEP           :    0
 PCIE_INFO: Number of RCEC            :    0
 PCIE_INFO: Number of EP              :    4
 PCIE_INFO: Number of RP              :    4
 PCIE_INFO: Number of iEP_EP          :    0
 PCIE_INFO: Number of iEP_RP          :    0
 PCIE_INFO: Number of UP of switch    :    0
 PCIE_INFO: Number of DP of switch    :    0
 PCIE_INFO: Number of PCI/PCIe Bridge :    0
 PCIE_INFO: Number of PCIe/PCI Bridge :    0
 Invalid IORT node type
 SMMU_INFO: Number of SMMU CTRL       :    2
 SMMU_INFO: SMMU index 00 version     :    v3.2
 SMMU_INFO: SMMU index 01 version     :    v3.2
 Peripheral: Num of USB controllers   :    10
 Peripheral: Num of SATA controllers  :    0
 Peripheral: Num of UART controllers  :    1
 SMBIOS: Num of slots :    1

      *** Starting PE tests ***
```

Figure 5-8: SBSA tests

```
Running debug dump
UEFI debug logs already run
Press any key to rerun UEFI debug logs
 ress any key within 1 seconds
Running BSA test
BSA Command String: bsa.efi -skip 830,900 -v 3
Press any key to start BSA in verbose mode.
If no key is pressed then BSA will be run in normal mode
BSA ACS is already run.seconds
Press any key to start BSA ACS execution from the beginning.
WARNING: Ensure you have backed up the existing logs.
 ress any key within 1 seconds
Running SBSA test
SBSA Command String: sbsa.efi -l 4 -skip 900 -v 3
Press any key to start SBSA in verbose mode.
If no key is pressed then SBSA will be run in normal mode
SBSA Command: sbsa.efi -l 4 -skip 900 -v 3


 SBSA Architecture Compliance Suite
    Version 7.2.2

 Starting tests for level  4 (Print level is  3)

 Creating Platform Information Tables
 PE_INFO: Number of PE detected       :    8
 Primary PE: MIDR_EL1                 :    0x410FD811
 GIC INFO: GIC version                :    v4
 GIC_INFO: Number of GICD             :    1
 GIC_INFO: Number of GICR RD          :    1
 GIC_INFO: Number of ITS              :    1
 TIMER_INFO: System Counter frequency :    1000 MHz
```

4. Boot ACS Linux, run Linux debug dump and FWTS tests:

Figure 5-9: ACS Linux boot

```
Running SBSA test
SBSA Command String: sbsa.efi -l 4 -skip 900 -v 3
Press any key to start SBSA in verbose mode.
If no key is pressed then SBSA will be run in normal mode
SBSA ACS is already run.econds
Press any key to start SBSA ACS execution from the beginning.
WARNING: Ensure you have backed up the existing logs.
 ress any key within 1 seconds
Booting Linux
EFI stub: Booting Linux Kernel...
EFI stub: Loaded initrd from command line option
EFI stub: ACS:acsforcevamap is 0
EFI stub: ACS:efi_novamap is 1
EFI stub: Generating empty DTB
EFI stub: Exiting boot services...
[    0.000000] Booting Linux on physical CPU 0x0000000a00 [0x410fd811]
[    0.000000] Linux version 6.10.0-00003-g2f165338a0b5 (cherat01@a080799) (aarc
h64-none-linux-gnu-gcc (Arm GNU Toolchain 13.2.rel1 (Build arm-13.7)) 13.2.1 202
31009, GNU ld (Arm GNU Toolchain 13.2.rel1 (Build arm-13.7)) 2.41.0.20231009) #1
 SMP PREEMPT Tue Apr 22 20:57:49 IST 2025
[    0.000000] KASLR enabled
[    0.000000] efi: Getting UEFI parameters from /chosen in DT:
[    0.000000] efi:   System Table         : 0x000000047ecd0018
[    0.000000] efi:   MemMap Address       : 0x0000000479b2c3c0
[    0.000000] efi:   MemMap Size          : 0x00000000000000ae0
[    0.000000] efi:   MemMap Desc. Size    : 0x0000000000000030
[    0.000000] efi:   MemMap Desc. Version : 0x0000000000000001
[    0.000000] efi: EFI v2.7 by EDK II
[    0.000000] efi: ACPI 2.0=0x479b20018 SMBIOS 3.0=0x47ea70000 TPMFinalLog=0x47
9af0000 MEMATTR=0x47ba71698 TPMEventLog=0x479b37018 INITRD=0x479b35018 RNG=0x479
b2df18 MEMRESERVE=0x479b35e98
[    0.000000] random: crng init done
```

Figure 5-10: Linux debug dump and FWTS tests

```
init.sh
Starting disk drivers
[   10.299826] nvme 0000:91:00.0: Adding to iommu group 0
[   10.305373] nvme nvme0: pci function 0000:91:00.0
[   10.316485] nvme nvme0: bad crto:0 cap:8202020ff0103ff
[   10.324399] nvme nvme0: missing or invalid SUBNQN field.
[   10.336343] nvme nvme0: allocated 40 MiB host memory buffer.
[   10.359829] nvme nvme0: 8/0/0 default/read/poll queues
[   10.378053]  nvme0n1: p1 p2 p3 p4
Mounted the results partition on device /dev/sda1
Collecting Linux Debug Dump
[   15.877254] pci 0000:01:00.0: invalid VPD tag 0x00 (size 0) at offset 0; assu
me missing optional EEPROM
[   15.893234] pci 0000:31:00.0: invalid VPD tag 0x00 (size 0) at offset 0; assu
me missing optional EEPROM
ping: bad address 'www.arm.com'
/
Linux Debug Dump - Completed
Executing FWTS for SBBR
FWTS Execution - Completed
```

5. Linux BSA and SBSA tests:

Figure 5-11: Linux BSA test

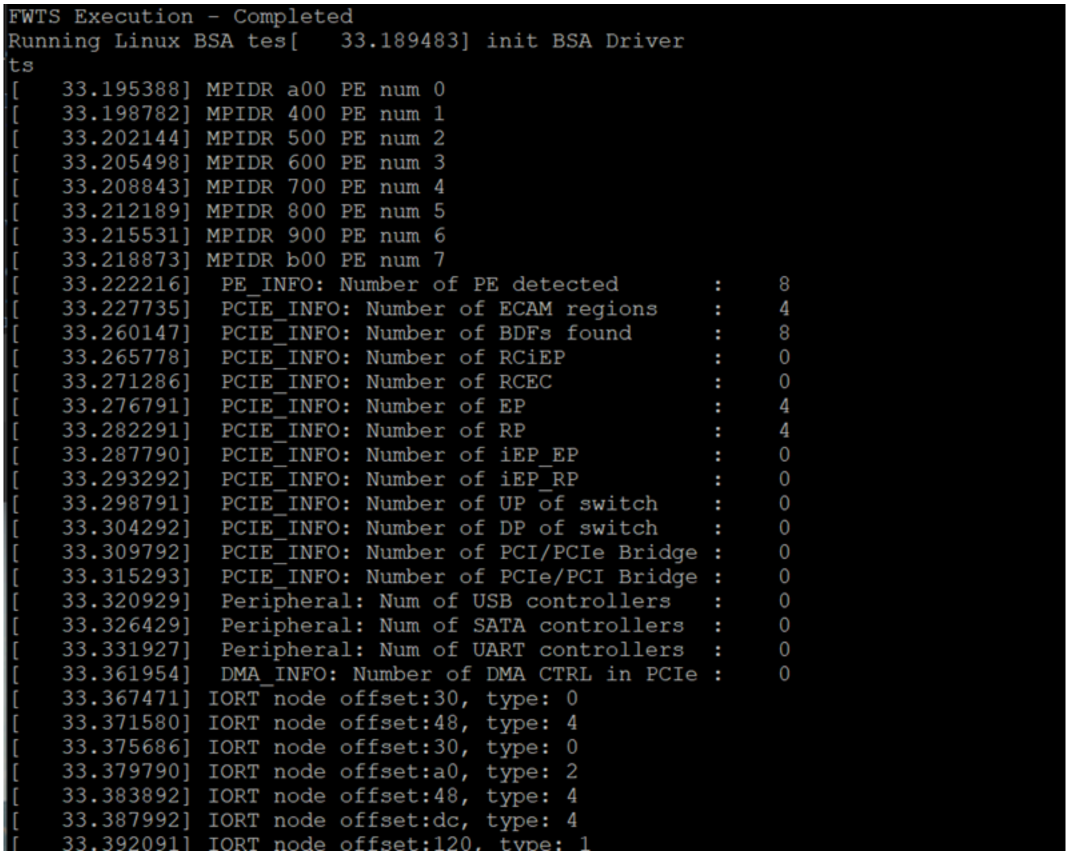


Figure 5-12: Linux SBSA test

```
Linux BSA test Execution - Completed
Running Lin[   39.472635] init SBSA Driver
ux SBSA tests
[   39.478135] MPIDR a00 PE num 0
[   39.481815] MPIDR 400 PE num 1
[   39.485121] MPIDR 500 PE num 2
[   39.488430] MPIDR 600 PE num 3
[   39.491740] MPIDR 700 PE num 4
[   39.495045] MPIDR 800 PE num 5
[   39.498364] MPIDR 900 PE num 6
[   39.501669] MPIDR b00 PE num 7
[   39.504969]  PE_INFO: Number of PE detected       :    8
[   39.510447]  PCIE_INFO: Number of ECAM regions    :    4
[   39.543377]  PCIE_INFO: Number of BDFs found      :    8
[   39.548968]  PCIE_INFO: Number of RCiEP           :    0
[   39.554445]  PCIE_INFO: Number of RCEC            :    0
[   39.559918]  PCIE_INFO: Number of EP              :    4
[   39.565387]  PCIE_INFO: Number of RP              :    4
[   39.570855]  PCIE_INFO: Number of iEP_EP          :    0
[   39.576323]  PCIE_INFO: Number of iEP_RP          :    0
[   39.581788]  PCIE_INFO: Number of UP of switch    :    0
[   39.587252]  PCIE_INFO: Number of DP of switch    :    0
[   39.592714]  PCIE_INFO: Number of PCI/PCIe Bridge :    0
[   39.598177]  PCIE_INFO: Number of PCIe/PCI Bridge :    0
[   39.603791]  Peripheral: Num of USB controllers   :    0
[   39.609254]  Peripheral: Num of SATA controllers  :    0
[   39.614713]  Peripheral: Num of UART controllers  :    0
[   39.647152] IORT node offset:30, type: 0
[   39.651231] IORT node offset:48, type: 4
[   39.655299] IORT node offset:30, type: 0
[   39.659366] IORT node offset:a0, type: 2
[   39.663430] IORT node offset:48, type: 4
```

After these tests are executed, a brief compliance summary and the message “ACS automated test suites run is completed.” is displayed and control returns to a Linux prompt.

Figure 5-13: End of automation test

```
Linux SBSA test Execution - Completed
Running edk2-test-parser tool
/
edk2-test-parser run completed
Running Device Driver Matching Script
/
Device Driver script run completed
Running acs log parser tool

WARNING: waiver.json ('/mnt/acs_tests/config/acs_waiver.json') must be provided to apply wa
ivers.
Waivers will not be applied.

WARNING: Log file /mnt/acs_results/bbsr/fwts/FWTSResults.log is missing.
WARNING: Log file /mnt/acs_results/bbsr/sct_results/Overall/Summary.log is missing.
WARNING: Log file /mnt/acs_results/bbsr/tpm2/verify_tpm_measurements.log is missing.
Suite: Recommended: BBSR-SCT: Not Compliant: not run
Suite: Recommended: BBSR-TPM: Not Compliant: not run
Suite: Mandatory  : FWTS: Compliant
Suite: Mandatory  : BSA: Compliant
Suite: Mandatory  : SBSA: Compliant
Suite: Recommended: BBSR-FWTS: Not Compliant: not run
Suite: Mandatory  : SCT: Compliant

SRS 3.0 Compliance result: Compliant

BBSR extension compliance results: Not Compliant (missing suite(s): BBSR-TPM, BBSR-FWTS, BB
SR-SCT)

ACS Merged JSON: /mnt/acs_results/acs_summary/acs_jsons/merged_results.json

ACS HTML Summary : /mnt/acs_results/acs_summary/html_detailed_summaries/acs_summary.html

Please wait acs results are syncing on storage medium.

ACS automated test suites run is completed.
Please reboot to run BBSR tests if not done
Please press <Enter> to continue ...
/ #
/ #
```

The following table shows more information about each test suite, including the running time details.

Table 5-1: Details about each test suite

Execution order	Test suite in ACS live image	Execution environment	Running time	Location of script and test suite	Notes
1	SCT	UEFI shell	1-6 hours Note: Using USB stick may make SCT tests faster, use a SATA or NVMe drive or an SSD in a USB enclosure	`acs_tests/bbr/SctStartup.nsh` `acs_tests/bbr/SCT/SCT.efi`	
2	UEFI debug dump	UEFI shell	Less than 3 minutes	`acs_tests/bsa/debug_dump.nsh`	
3	BSA (UEFI)	UEFI shell	Less than 5 minutes	`acs_tests/bsa/bsa.nsh` `acs_tests/bsa/bsa.efi`	
4	SBSA (UEFI)	UEFI shell	Less than 3 minutes	`acs_tests/bsa/sbsa/sbsa.nsh` `acs_tests/bsa/sbsa/sbsa.efi`	Only run if automation_sbsa_run in acs_run_config.ini is set to true.

Execution order	Test suite in ACS live image	Execution environment	Running time	Location of script and test suite	Notes
5	Linux debug dump	ACS Linux	Less than 3 minutes	In `ramdisk-buildroot.img Init.sh`	
6	FWTS	ACS Linux	Less than 3 minutes	In `ramdisk-buildroot.img Init.sh /bin/fwts`	
7	BSA (Linux)	ACS Linux	Less than 1 minute	In `ramdisk-buildroot.img Init.sh /lib/modules/bsa_acs.ko`	
8	SBSA (Linux)	ACS Linux	Less than 1 minute	In `ramdisk-buildroot.img Init.sh /lib/modules/sbsa_acs.ko`	Only run if automation_sbsa_run in acs_run_config.ini is set to true.

5.8 Run tests in normal mode

When the image boots, choose one of the following options on the ACS grub menu for your manual test:

- Linux boot to boot ACS Linux and execute FWTS and Linux BSA with the same commands as automation test.
- SystemReady band ACS (Automation) to execute the tests in the same sequence as fully automated mode. Note that you can customize automation tests by modifying `acs_tests/config/acs_run_config.ini` and then use this option to launch your customized automation tests.
- BBSR compliance (Automation) to start the BBSR automation test. Note that you need to enroll the test secure boot keys in the ACS image before launching the automation test. For details, refer to the "How do I test for Base Boot Security Requirements (BBSR) compliance" section in System FAQ.
- UEFI Execution Environment to boot UEFI shell without running any scripts and tests. For details, please see Running ACS tests manually.
- Linux Execution Environment to boot ACS Linux without running any scripts and tests.
- Linux Boot with SetVirtualAddressMap enabled to test firmware's SetVirtualAddressMap implementation.

Figure 5-14: Options on ACS grub menu

Note

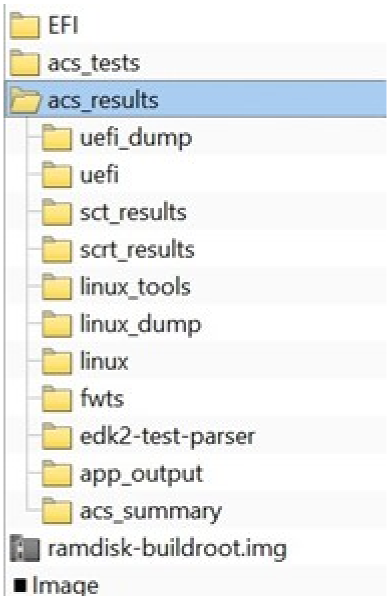
You can also skip individual test stages by pressing a key at the appropriate point.

5.9 Review test results and determine compliance

After testing, check the files in the `acs_results` folder to ensure that all necessary logs are generated.

Check that the folders shown in the following figure are generated:

Figure 5-15: ACS results directory



To check the test result, you can either directly navigate to /mnt/acs_results in ACS Linux to view the logs or extract the logs from the ACS drive to view the logs on the host machine.

Review the test result and determine the compliance

The first step is to review the summary and find out the failed tests (failures). There are two ways to review the summary:

1. Check the brief summary which is shown on the screen at the end of testing. In the brief summary, you can get the name of test suite and the number of failures from the message with the format: "Suite: Mandatory : <test suite name> : Not Compliant : Fail <Number of failures>".
2. Check the ACS summary report file located at acs_results/acs_summary/html_detailed_summaries/acs_summary.html. In the summary report, you can get the name of test suite and the number of failures by searching for the keyword "Failed".

After getting the name of test suite and the number of failures, you can use the following table to find out the corresponding report and log files and get more information about failures. Then, use that information to fix the compliance issues.

Note that some failed tests may not be compliance issues because they may be based on some conditional requirements in the SystemReady specifications. To identify those non-critical failed tests, refer to the following sections in the SystemReady FAQ:

- How can partners independently evaluate test results and determine compliance?
- How can partners decide whether a test failure is critical for self-compliance?
- Appendix A - Potentially waivable failures for SystemReady band compliance

Table 5-2: Location of reports and logs

Test suites	Location of the report	Location of the log
BSA (UEFI)	`acs_results/acs_summary/html_detailed_summaries/bsa_detailed.html`	`acs_results/uefi/BsaResults.log`
BSA (Linux)	`acs_results/acs_summary/html_detailed_summaries/bsa_detailed.html`	`acs_results/linux/BsaResultsKernel.log`
SBSA (UEFI)	`acs_results/acs_summary/html_detailed_summaries/sbsa_detailed.html`	`acs_results/uefi/SbsaResults.log`
SBSA (Linux)	`acs_results/acs_summary/html_detailed_summaries/sbsa_detailed.html`	`acs_results/linux/SbsaResultsKernel.log`
SCT (UEFI)	`acs_results/acs_summary/html_detailed_summaries/sct_detailed.html`	`acs_results/edk2-test-parser/edk2-test-parser.log` and `acs_results/sct_results/Overall/Summary.logs`
FWTS (Linux)	`acs_results/acs_summary/html_detailed_summaries/fwts_detailed.html`	`acs_results/fwts/results.log`
Debug Dumps	N/A	`acs_results/uefi_dumps` `acs_results/linux_dumps`

In Log files, you can use the keywords below to find the failures:

- For BSA and SBSA test failures, search for the keyword “Result: FAIL” to find more information about the failures
- For SCT test failures, search for the keyword “[FAILED]” and “FAILURE” to find more information about the failures
- For FWTS test failures, search for the keyword “Test Failure Summary” or “FAILED [” to find more information about the failures

For the location of reports in HTML format and the logs files that are generated by ACS, refer to that table above.

Determine the compliance

After clarifying failed tests and fixing compliance issues, the system should be compliant with no failure or a few non-critical (waivable) failures. Then, partners can perform self-declared compliance with the SystemReady band wordmark without Arm's permission. For example, partners can declare “our device XYZ is SystemReady Band v3.0 compliant”.

For more information about performing self-declared compliance, please refer to the questions in the section “SystemReady Band – General FAQ” in the SystemReady FAQ.

6. OS testing preparation

Note that OS testing is not required for SystemReady band compliance. However, it is recommended to run the OS tests to further validate compliance in addition to ACS live image testing. This section describes how to prepare for OS testing.

6.1 Prepare the OS installer media

Operating systems used for compliance testing must boot free of board-specific images and with generic installation instructions. For example, do not use versions of an operating system or installation guides that are specifically designed for Raspberry Pi. The SystemReady band does not use special images and guides, and ensures your images are suitable for AARCH64.

Before you prepare the installer media, download the AARCH64 installer image for your OS. The following table provides links to install tested OSes for the System Ready band. For more information, see OS-image-download-links.txt in the band template.

Table 6-1: Operating System Download Links

Operating system	Download link
Red Hat Enterprise Linux (RHEL)	RHEL Server ISO - RHEL ARM 64
Fedora Server	Standard ISO image for aarch64
Fedora Workstation (Live ISO)	aarch64 Live ISO
SUSE Linux Enterprise Server (SLES)	Evaluation Copy of SUSE Linux Enterprise Server | SUSE
OpenSUSE Leap	OpenSUSE DVD iso
OpenSUSE Tumbleweed (Daily Build)	openSUSE Tumbleweed - Get openSUSE
Ubuntu Server	64-bit ARM (ARMv8/AArch64) server install image
Ubuntu Desktop Live (Daily Build)	64-bit ARM (ARMv8/AArch64) desktop image
Debian	arm64 DVD iso
NetBSD	NetBSD/evbarm
OpenBSD	OpenBSD FAQ: Installation Guide
FreeBSD	Download FreeBSD | The FreeBSD Project
VMware ESXi-Arm Fling	ESXi Arm Edition

Note

Entry in this list does not indicate that the OS is officially supported on the system. Consult the system and OS vendors for official support.

To set up an OS installer USB drive:

1. Insert the USB drive then use a disk tool to write an OS installer image to the drive.
    - Rufus or balenaEtcher on Windows

    - Use balenaEtcher for RHEL, Fedora, CentOS, and AlmaLinux because of an OS installer known issue that results in a “source can't be found” error.
- `dd` command on Linux

   For example, if your USB drive is `/dev/sda` and you want to write the Ubuntu install image, use the following command:

```
dd if=ubuntu-22.04.4-live-server-arm64.iso of=/dev/sda status=progress
```

2. If installation problems occur, for example a system hang in the OS bootloader, clean the media device as follows:
   - `diskpart` on Windows:

```
C:\diskpart
DISKPART> list disk
DISKPART> select disk x (Where “x” it's the letter of the USB drive)
DISKPART> clean (if installation issue still exists, try “clean all” This may
 take hours.)
DISKPART> exit
```

   - `dd` command on Linux:

   First, only clean the first megabyte. In most cases, this fixes the issue:

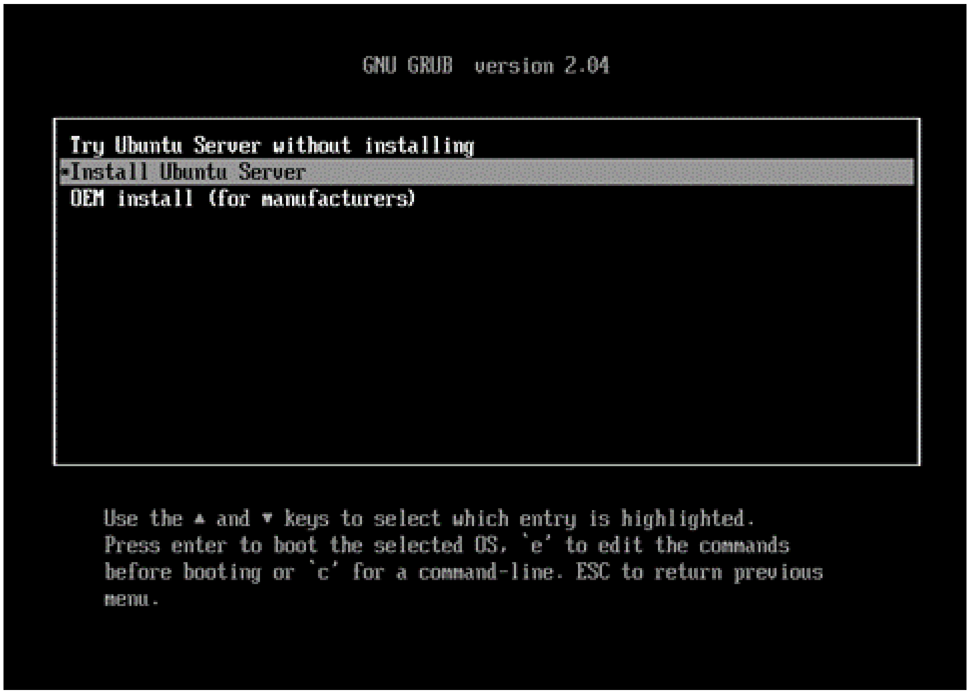
```
dd if=/dev/zero of=/dev/sdb bs=1M count=1 status=progress
```

   If the installation issue persists, perform a deep clean. This might take several hours:

```
dd if=/dev/zero of=/dev/sdb bs=1M status=progress
```

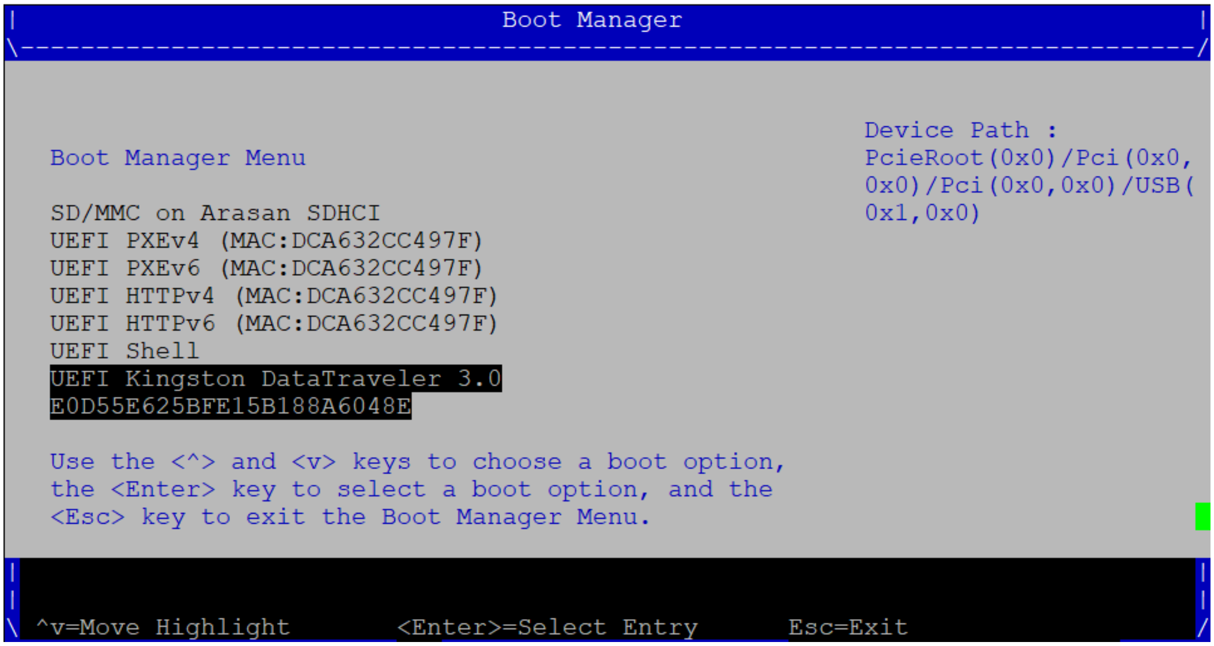
3. After you create the install media, insert the drive into the USB 3.0 (blue) USB ports on the system.
4. If the USB drive is the first boot option, UEFI discovers and automatically boots into the installer media. Figure 6-1: GRUB loader on page 41 shows the OS bootloader:

Figure 6-1: GRUB loader



5. If the first boot option is UEFI shell or PXE boot, press Esc to interrupt the boot process.
6. In the UEFI menu, go to **Boot Manager**
7. Choose the install media (USB drive).
8. Figure 6-2: USB key in Boot Manager on page 42 shows the USB key which is called UEFI Kingston DataTraveler 3.0:

Figure 6-2: USB key in Boot Manager



9. Press **Enter** to launch the OS bootloader.
10. Now, you can follow the installation instructions provided by your OS. For example, see Ubuntu or Fedora.
11. Install the operating system to a storage device, not the installer media or the SD card that you used to store your firmware.

Note

Many operating systems have images and guides specific to a platform like Raspberry Pi 4. However, these guides are often designed without SystemReady compliance considerations.

VMware offers ESXi-Arm Fling as a technical preview for evaluation. For more information, see ESXi Arm Edition.

6.2 Boot order

When UEFI variables are not supported at runtime, the OS might not be able to create a boot entry. The installed OS might not be automatically booted after installation and reboot.

In this case, you can modify the boot order to solve this problem:

1. After installation, power cycle the system an extra time or enter the UEFI configurator as described in Configure UEFI.
2. Open the Boot Maintenance Manager and change the boot order.

3. The installed OS device must be at the top of the list. If it is not, highlight the device and press + until it is at the top of the list.
4. Press **Enter**, then save and exit.

7. Windows OS testing

Although Windows OS testing is not required for SystemReady band compliance, it is still recommended to further validate compliance in addition to ACS live image testing.

There are two types of Windows OS that can be used for testing:

- Windows 11
- WinPE (Windows Preinstallation Environment).

WinPE is a lightweight version of Windows that can be used for testing basic Windows boot and functions. Windows 11 can then be used to troubleshooting any issues that arise.

For testing Windows 11, you can download the installer image (.iso) from https://www.microsoft.com/en-us/software-download/windows11arm64, and then follow the instructions in Prepare the OS installer media to deploy the installer image to the a USB drive, and then boot from that USB drive for installation.

For testing WinPE, there is no installer or live image `.iso` that can be downloaded from the Microsoft website The following sections describe how to build and configure the image for your testing.

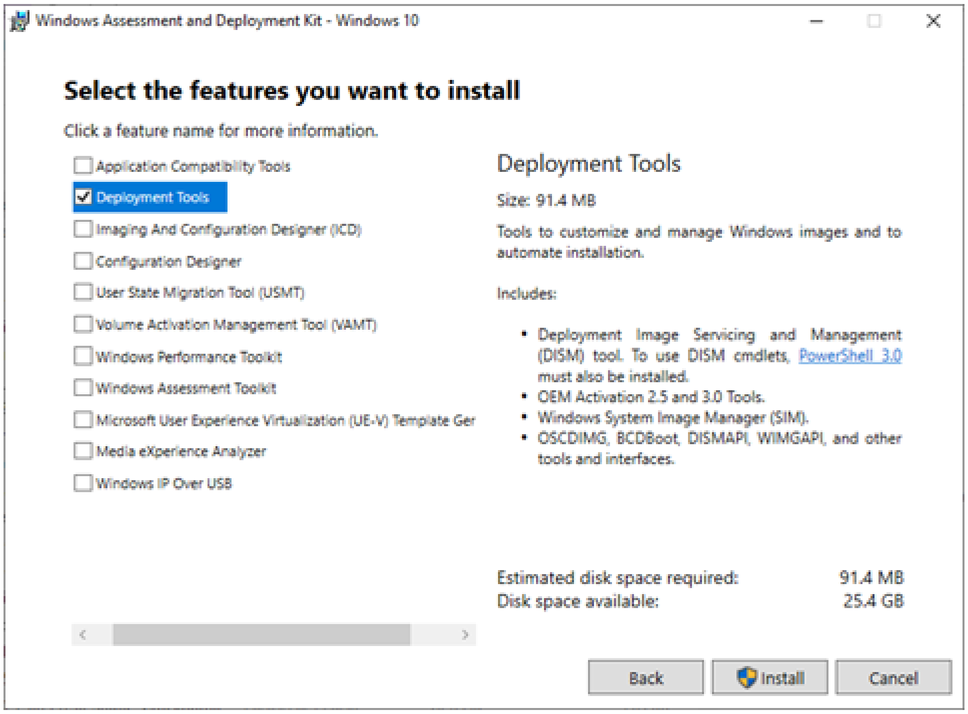
7.1 Download and run Windows ADK and WinPE

Microsoft does not provide a `.iso` file for WinPE. Instead, download the Windows ADK and Windows PE add-on tools from https://docs.microsoft.com/en-us/windows-hardware/get-started/adk-install to build one yourself. If you want to build a WinPE image based on the latest Windows kernel, download the insider preview version of tools from https://www.microsoft.com/en-us/software-download/windowsinsiderpreviewADK.

Follow the steps below to install and run two tools: Windows Assessment and Deployment Kit (Windows ADK) and Windows PE add-on.

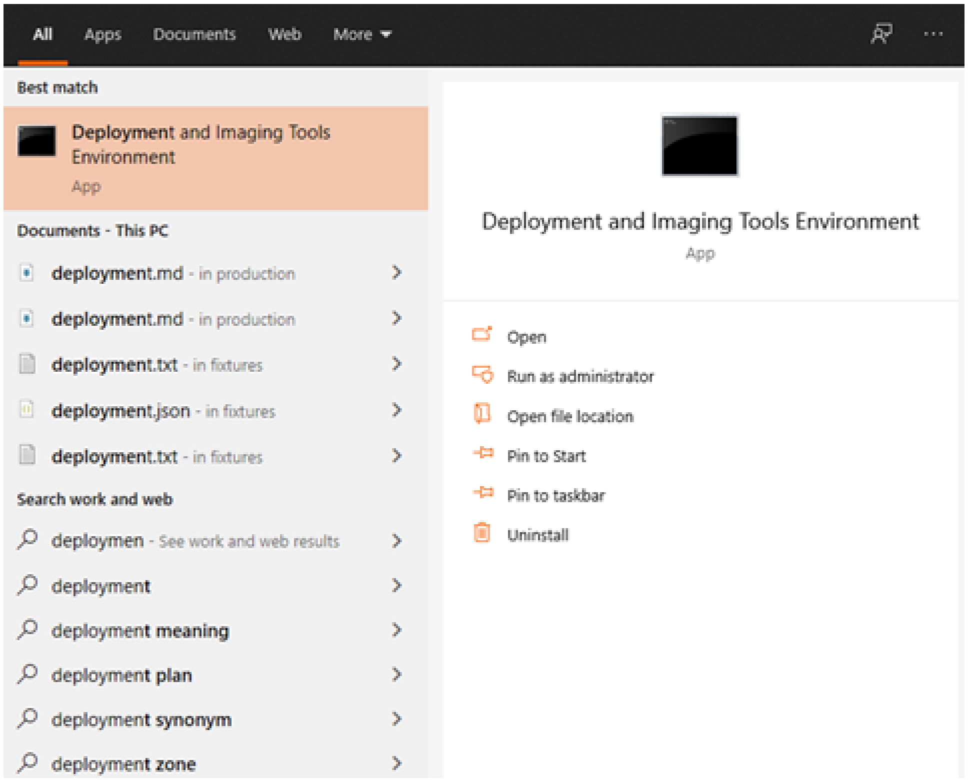
1. Run the Windows ADK installer `adksetup.exe`.
2. Select **Install the Windows Assessment and Deployment Kit – Windows 10 to this Computer** and follow the installer to feature selection.
3. Figure 7-1: Windows ADK features on page 45 shows enabling the **Deployment Tools** feature to build a WinPE image:

Figure 7-1: Windows ADK features



4. Run the WinPE add-on tool installer `adkwinpesetup.exe` and install the Windows Preinstallation Environment feature.
5. Create a bootable WinPE USB drive using the **Deployment and Imaging Tools Environment** as Administrator. Figure 7-2: Starting Deployment and Imaging Tools Environment on page 46 shows how to start the Deployment and Imaging Tools Environment app window with administrator privileges:

Figure 7-2: Starting Deployment and Imaging Tools Environment



The Create bootable WinPE media guide uses amd64 architecture. Use Arm64 architecture to build an Arm64 USB.

6. If you are creating an ISO file, follow the instructions in Create an ISO file to change the boot parameters.
7. Run the following command to create a working copy of the Windows PE arm64 files:

```
> copype arm64 C:\WinPE_arm64
```

7.2 Create an ISO file

To create an ISO file:

1. The files in the `\media` folder are copied to the USB key. This lets you change the boot parameters without having to mount the ISO.
2. To output only OS messages without debug messages to the serial console, use the commands below to enable OS messages and disable debug messages on the serial console:

```
> cd C:\WinPE_arm64\media\EFI\Microsoft\Boot

C:\WinPE_arm64\media\EFI\Microsoft\Boot> bcdedit /store BCD /set {default} ems ON
```

```
C:\WinPE_arm64\media\EFI\Microsoft\Boot> bcdedit /store BCD /set {default} debug
 off
```

3. Use the following command to create the ISO image.

```
> MakeWinPEMedia /ISO C:\WinPE_arm64 C:\WinPE_arm64\WinPE_arm64.iso
```

7.3 Create a WinPE USB drive

To write a WinPE iso image to a USB drive, you can use tools like Rufus in Windows OS or WoeUSB in Linux OS.

If you want to directly generate a WinPE USB drive, use the following commands:

1. Clean a selected USB drive, create a primary partition, format it to FAT32, assign it the letter P, and label it "WINPE".

```
C:\diskpart
DISKPART> list disk
DISKPART> select disk x (Where "x" it's the number of the USB drive)
DISKPART> clean (if installation issue still exists, try "clean all" This may
 take hours.)
DISKPART> create partition primary
DISKPART> format fs=fat32 quick label="WINPE"
DISKPART> assign letter P
DISKPART> exit
```

2. To install directly to the USB drive and format the drive, use the following command:

```
> MakeWinPEMedia /UFD C:\WinPE_arm64 P:
```

3. To enable the EMS serial console on the WinPE media, enter the following commands:

```
> P:
P:\> cd P:\EFI\Microsoft\Boot\
P:\EFI\Microsoft\Boot> bcdedit /store BCD /set {default} ems ON
```

7.4 Other Boot Configuration Data settings

If the system has one UART, you cannot enable WinDBG and EMS at the same time.

1. To enable WinDBG serial debug, use the following commands:

```
> bcdedit /store BCD /dbgsettings SERIAL DEBUGPORT:1 BAUDRATE:115200
> bcdedit /store BCD /set {default} debug ON
```

2. Enter `bcdedit /store BCD /enum all` to list all Boot Configuration Data (BCD) settings.

7.5 Boot WinPE

To boot WinPE on an Arm64 system:

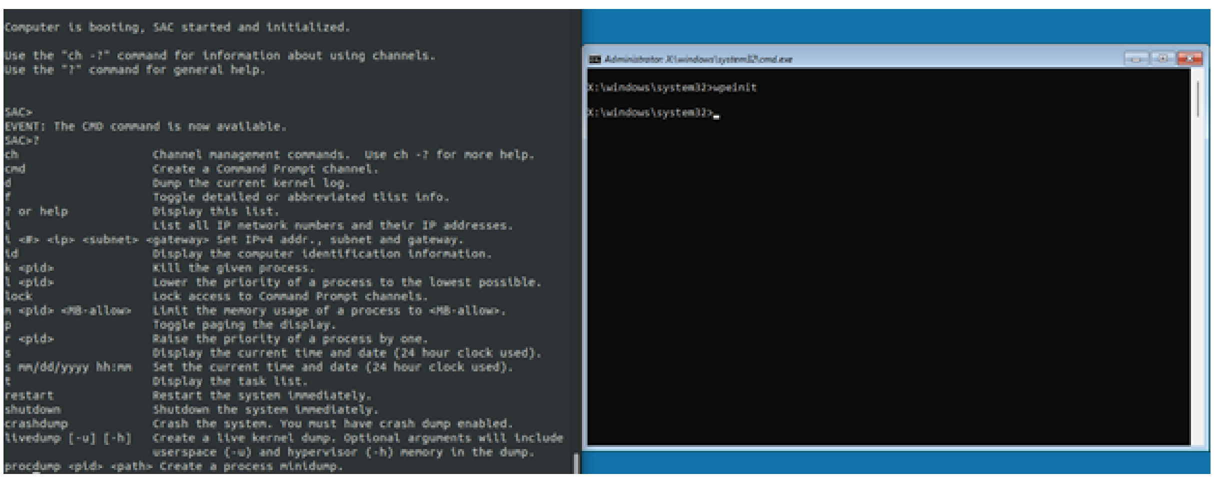
1. Flash the WinPE ISO image to a media device, for example a USB drive.
2. Install the media device on the system, for example by plugging the USB drive into a USB port
3. Boot from the media device.

Note

If you do not have an Arm64 system, use Arm Infrastructure FVPs.

1. Press any key to boot WinPE from CDROM. A `cmd` window is displayed and a SAC console in the UART terminal if you enabled EMS in the boot configuration. Figure 7-3: SAC console and cmd window on page 48 shows an example of the console and cmd window:

Figure 7-3: SAC console and cmd window



8. Integrate compliance tests into production processes

There are two ways to integrate and run compliance tests in production processes:

1. Integrate the ACS live image into your production release test.
2. Integrate individual binaries and test suites from the ACS live image into your production release test.

8.1 Integrate ACS live image into the production release test

To integrate the ACS live image into your tests, do the following:

1. Make a preparation script to perform the following steps:
    - Identify a non-system, removable, or unmounted drive. For Linux, you can use the `lsblk` command.
    - Download the latest ACS live image from https://github.com/ARM-software/arm-systemready/releases/download/. For Linux, you can use the `wget` command as follows:

      ```
      wget https://github.com/ARM-software/arm-systemready/releases/download/
      v25.04_SR_3.0.1/systemready_acs_live_image.img.xz
      ```

    - Decompress the downloaded file and write the image to the non-system drive. For Linux, you can use the `xz` and `dd` commands as follows.

   ```
   xz -d systemready_acs_live_image.img.xz
   dd if=systemready_acs_live_image.img of=/dev/<NON-SYSTEM DRIVE>
   ```

- Move the boot option of the non-system drive to the top of the UEFI boot order. For Linux, you can use the `efibootmgr` command.
- Reboot the system to start the compliance automation test.

1. Add a line to load the preparation script in your test script that originally runs the tests for your product release.

8.2 Integrate individual binaries/test suites from ACS live image into your test

The following information shows how to integrate each test suite from the ACS live image into your test process.

SCT

The SCT test suite in the ACS live image uses the UEFI shell execution environment.

To integrate the test suite, do the following:

1. Copy the `acs_tests/bbr/SCT/` folder from the ACS drive to the FAT32 partition that you used to put your UEFI test suites (`.efi`) for your production testing.
2. Add the following two commands to your test script to run the SCT test and resume the SCT test after reboot:

   ```
   SCT.efi -s SBBR.seq
   SCT.efi -c
   ```

   For more information, refer to the ACS SCT script `acs_tests/bbr/SctStartup.nsh`.

   Note that if you have already used the SCT in production testing, you can merge the test selection settings from SBBR.seq to your sequence file using the `SCT -u command`.

BSA (UEFI)

The BSA (UEFI) test suite in the ACS live image uses the UEFI shell execution environment.

To integrate the test suite, do the following:

1. Copy `bsa.efi` either from the `/acs_tests/bsa folder` in the ACS drive or from bsa-acs prebuilt images to the FAT32 partition that you used to put your UEFI test suites (`.efi`) for your production testing. If you would like to build an image from the latest source code, refer to ACS build steps - UEFI Shell application
2. Add the following command to your test script to run the BSA test:

   ```
   bsa.efi -skip 900 -f BsaResults.log
   ```

   For more information, refer to the ACS SCT script `acs_tests/bsa/bsa.nsh`.

SBSA (UEFI)

The SBSA (UEFI) test suite in the ACS live image uses the UEFI shell execution environment.

To integrate the test suite, do the following:

1. Copy `sbsa.efi` either from the `/acs_tests/bsa/sbsa` folder in ACS drive or from sbsa-acs prebuilt images to the FAT32 partition that you used to put your UEFI test suites (`.efi`) for your production testing. If you would like to build an image from the latest source code, refer to ACS build steps - UEFI Shell application.

2. Add the following command to your test script to run the SBSA test.

```
sbsa.efi -skip 900 -f SbsaResults.log
```

For more information, refer to the ACS SBSA script `acs_tests/bsa/sbsa/sbsa.nsh`.

FWTS

The SBSA (UEFI) test suite in the ACS live image uses the ACS Linux execution environment.

To integrate the test suite, do the following:

1. Add commands to install FWTS in your Linux distro that you used to run production testing. For Ubuntu, you can use the following commands to install FWTS:

```
sudo apt update
sudo apt upgrade --assume-yes
sudo apt install --assume-yes software-properties-common
sudo add-apt-repository ppa:firmware-testing-team/ppa-fwts-stable
sudo apt-get update
sudo apt-get install fwts
```

2. Add the following command to your test script to run the FWTS test:

```
fwts -r stdout -q --uefi-set-var-multiple=1 --uefi-get-mn-count-multiple=1 --sbbr
 esrt uefibootpath aest cedt slit srat hmat pcct pdtt bgrt bert einj erst hest
 sdei nfit iort mpam ibft ras2
```

Note that If you have already used the FWTS in production testing, you can simply review the options used in step 2 above and append the missing options to your `fwts` command.

BSA (Linux)

The BSA (Linux) test suite in the ACS live image uses the ACS Linux execution environment.

To integrate the test suite, do the following:

1. Download the source code of BSA ACS Linux kernel module `bsa_acs.ko` and rebuild it in your Linux distro. For details, refer to ACS build steps - Linux application.
2. Add the following commands to your test script to load the BSA ACS Linux kernel module and run the Linux BSA test:

```
insmod bsa_acs.ko
./bsa
```

SBSA (Linux)

The SBSA (Linux) test suite in the ACS live image uses the ACS Linux execution environment.

To integrate the test suite, do the following:

1. Download the source code of SBSA ACS Linux kernel module `bsa_acs.ko` and rebuild it in your Linux distro. For details, refer to ACS build steps - Linux application.

2. Add the following commands to your test script to load the SBSA ACS Linux kernel module and run the Linux BSA test:

```
insmod sbsa_acs.ko
./sbsa
```

9. Debugging commands

This section describes some useful commands that are helpful for debugging compliance issues.

For the UEFI shell commands that are helpful for debugging compliance issues, refer to ACS UEFI shell debug dump script.

Table 9-1: Debugging commands on page 53 shows some Linux commands that are helpful for debugging compliance issues. For more commands, see ACS Linux debug dump script.

Table 9-1: Debugging commands

Command	Description
`hostnamectl`	Control the system hostname
`lspci`	Display information about PCI buses in the system and devices connected to them
`lspci -vvv`	Display everything that can be parsed
`lsusb`	Display information about USB buses in the system and the devices connected to them
`lsusb -v`	Display detailed information about the USB devices shown. This information includes configuration descriptors for the current speed of the device. Class descriptors are shown for USB device classes including hub, audio, HID, communications, and chipcard.
`df`	Report file system disk space usage
`cat /etc/os-release`	Show operating system identification data

For the Windows commands that are helpful for debugging compliance issues, refer to the WinPE commands in compliance test result template readme.

10. ACPI requirements and implementations

SystemReady band compliant devices must be conformant with the following specifications:

- BSA
- SBBR recipe in BBR
- SBSA (only for servers)

The Advanced Configuration and Power Interface (ACPI) describes the hardware resources that are installed on SystemReady band compliant devices. ACPI also handles aspects of runtime system configuration, event notification, and power management.

For mandatory ACPI tables for SystemReady band compliant systems, see the Arm Base Boot Requirement (BBR) specification. For example, the Raspberry Pi 4 uses the following mandatory ACPI tables:

- Root System Description Pointer (RSDP)
- Extended system Description Table (XSDT)
- Fixed ACPI Description Table (FACP)
- Differentiated System Description Table (DSDT)
- Debug Port 2 Table (DBG2)
- Generic Timer Descriptor Table (GTDT)
- Multiple APIC Description Table (APIC)
- Processor Property Topology Table (PPTT)
- SPCR Serial Port Console Redirection Table. This table is not published by default. To publish this table, select Device Manager in the UEFI menu, then select Serial as the console device.
- Secondary System Description Table (SSDT)
- PCI Memory-mapped Configuration Space (MCFG)

The ACPI examples in this section demonstrate the following use cases:

- Thermal zones
- Fan cooling devices
- USB XHCI and PCIe
- UART
- Debug port
- Power buttons
- PCIe ECAM

10.1 Example: Thermal zone

Raspberry Pi 4 has hardware resources that allow the OS to perform thermal management. BCM2711 provides a register to read CPU temperature. You can enable platform-specific hardware resources by exposing memory map peripheral addresses with Devicetree or ACPI structures, and provide platform-specific OS drivers. For example, the bcm2711_thermal Linux driver consumes a register address provided through a Devicetree structure and produces an API to read CPU temperature. The OS requires an update for any hardware modifications because a new driver is installed to control this hardware. We recommend that you abstract these hardware resources using ACPI AML methods. In this example, you do not use a platform driver because the hardware resource is represented as an ACPI thermal model.

The following table defines a simple thermal zone TZ00. TZ00 specifies the following methods:

Table 10-1: ACPI Methods

Method	Description
`_TMP`	Returns the thermal zone's current temperature in tenths of degrees
`_SCP`	Sets the platform cooling policy, active or passive. A placeholder on the Raspberry Pi.
`_CRT`	Returns the critical trip point in tenth of degrees where OSPM must perform a critical shutdown
`_HOT`	Returns the critical trip point in tenths of degrees where OSPM can choose to transition the system into S4 sleeping state
`_PSV`	Return the passive cooling policy threshold value in tenths of degrees

The following objects are also presented:

Table 10-2: ACPI Objects

Object	Description
`_TZP`	Thermal zone polling frequency in tenths of seconds
`_PSL`	List of processor device objects for clock throttling. Specifies all four cores on Raspberry Pi.

The following code shows a thermal zone (TZ00) implementation:

```
    Device (EC00)
    {
      Name (_HID, EISAID ("PNP0C06"))
      Name (_CCA, 0x0)

      // all temps in are tenths of K (aka 2732 is the min temps in Linux (aka 0C))
      ThermalZone (TZ00) {
        Method (_TMP, 0, Serialized) {
          OperationRegion (TEMS, SystemMemory, THERM_SENSOR, 0x8)
          Field (TEMS, DWordAcc, NoLock, Preserve) {
            TMPS, 32
          }
          return (((410040 - ((TMPS & 0x3ff) * 487)) / 100) + 2732);
        }
        Method (_SCP, 3) { }                // receive cooling policy from OS

        Method (_CRT) { Return (3632) }     // (90C) Critical temp point (immediate
power-off)
        Method (_HOT) { Return (3582) }     // (85C) HOT state where OS should
hibernate
```

```
        Method (_PSV) { Return (3532) }    // (80C) Passive cooling (CPU throttling)
trip point

        // SSDT inserts _AC0/_AL0 @60C here, if a FAN is configured

        Name (_TZP, 10)                   //The OSPM must poll this device every 1
seconds
        Name (_PSL, Package () { \_SB_.CPU0, \_SB_.CPU1, \_SB_.CPU2, \_SB_.CPU3 })
      }
    }
```

10.2 Example: Fan cooling device

Raspberry Pi 4 can be connected to extension hats with a variable speed fan, such as a POE hat. You can also connect a simple on/off fan. A POE hat uses the Raspberry Pi 4 proprietary mailbox for fan control and an on/off fan can be controlled with a single GPIO pin. As a result, each fan device uses specific drivers and can be presented to the OS in different ways.

To simplify OSPM and remove the platform driver, ACPI objects and methods can provide fan device information and control to the OS.

ACPI 1.0 defines a fan device, which is suitable for an on/off fan connected to GPIO. ACPI 4.0 defines additional fan device interface objects, enabling OSPM to perform more robust active cooling thermal control.

Currently, Raspberry Pi 4 supports the ACPI 1.0 fan device.

The following table lists the PFAN fan power resource methods:

Table 10-3: PFAN Fan Power Resource methods

Method	Description
`_STA`	Returns the status of a fan device. This example returns the exact value of the GPIO pin which is used to connect a fan. The exact pin used is configured in the UEFI menu.
`_ON`	Puts the power resource into ON state by setting the GPIO pin, which is used to control a fan
`_OFF`	Puts the power resource into OFF state by clearing the GPIO pin, which is used to connect a fan

The following table lists methods and objects for the fan device:

Table 10-4: Fan Device Methods and Objects

Object	Description
`FAN0`	Fan device object
`_HID`	Plug and Play ID. This should be PNP0C0B
`_PR0`	Power Resource for the fan object (fully ON state)

The following table lists methods and objects for the Active Cooling point:

Table 10-5: Active Cooling Point Methods and Objects

Object	Description
`_AC0`	Returns the temperature trip point at which OSPM must start or stop Active cooling
`_AL0`	Evaluates a list of Active cooling devices to be turned on when the corresponding `_ACx` temperature threshold is exceeded. `_AL0` defines a single FAN0 device on RPi4

The following code shows the ACPI implementation of a fan cooling device and the device resources:

```
Scope (\_SB_.EC00)
  {
    // Define a NameOp we will modify during InstallTable
    Name (GIOP, 0x2) //08 47 49 4f 50 0a 02 (value must be >1)
    Name (FTMP, 0x2)
    // Describe a fan
    PowerResource (PFAN, 0, 0) {
      OperationRegion (GPIO, SystemMemory, GPIO_BASE_ADDRESS, 0x1000)
      Field (GPIO, DWordAcc, NoLock, Preserve) {
        Offset (0x1C),
        GPS0, 32,
        GPS1, 32,
        RES1, 32,
        GPC0, 32,
        GPC1, 32,
        RES2, 32,
        GPL1, 32,
        GPL2, 32
      }
      // We are hitting a GPIO pin to on/off a fan.
      // This assumes that UEFI has programmed the
      // direction as OUT. Given the current limitations
      // on the GPIO pins, its recommended to use
      // the GPIO to switch a larger voltage/current
      // for the fan rather than driving it directly.
      Method (_STA) {
        if (GPL1 & (1 << GIOP)) {
          Return (1)                 // present and enabled
        }
        Return (0)
      }
      Method (_ON)  {                // turn fan on
        Store (1 << GIOP, GPS0)
      }
      Method (_OFF) {                // turn fan off
        Store (1 << GIOP, GPC0)
      }
    }
    Device (FAN0) {
      // Note, not currently an ACPIv4 fan
      // the latter adds speed control/detection
      // but in the case of linux needs FIF, FPS, FSL, and FST
      Name (_HID, EISAID ("PNP0C0B"))
      Name (_PR0, Package () { PFAN })
    }
  }
  // merge in an active cooling point.
  Scope (\_SB_.EC00.TZ00)
  {
    Method (_AC0) { Return ( (FTMP * 10) + 2732) } // (60C) active cooling trip
 point,
                                                   // if this is lower than PSV then
 we
                                                   // prefer active cooling
    Name (_AL0, Package () { \_SB_.EC00.FAN0 })    // the fan used for AC0 above
  }
```

With the ACPI 1.0 fan, you do not need a platform-specific GPIO driver and a temperature monitor. The ACPI fan driver consumes the PNP0C0B FAN0 device and uses an ACPI power subsystem to turn it on or off.

Use the following Hat 4 methods with ACPI 4.0 on a Raspberry Pi 4 for POE:

Table 10-6: Hat 4 Methods

Method	Description
`_FIF`	Returns fan device information
`_FPS`	Returns a list of supported fan performance states
`_FSL`	Control method that sets the fan device's speed level (performance state). RPI_FIRMWARE_SET_POE_HAT_VAL would be used in ACPI AML on a Raspberry Pi 4.

In this example, instead of exposing a proprietary mailbox to the OS and using a platform driver, we allow the OS to use a standard ACP fan driver.

10.3 Example: Non-PCIe USB XHCI

If a BSA compliant PCIe USB controller is present and visible by the operating system, you must use an ACPI MCFG table.

The PCIe controller is present on the Raspberry Pi 4, but it is not BSA compatible. To work around the compliance issue, the PCIe is hidden and as a result MCFG is not used.

Instead, an ACPI node XHC0 is added to the DSDT table. Also, a _DMA object is defined to describe resources consumed by XCH0.

The following code shows the ACPI_DMA resource:

```
Name (_DMA, ResourceTemplate() {
        /*
         * XHC0 is limited to DMA to first 3GB. Note this
         * only applies to PCIe, not GENET or other devices
         * next to the A72.
         */
        QWordMemory (ResourceConsumer,+
            ,
            MinFixed,
            MaxFixed,
            NonCacheable,
            ReadWrite,
            0x0,
            0x0,        // MIN
            0xbfffffff, // MAX
            0x0,        // TRA
            0xc0000000, // LEN
            ,
            ,
            )
     })
```

`_DMA` is an optional object and returns a byte stream in the same format as a `_CRS` object. `_DMA` is defined under devices that represent buses, such as Device SCB0 for the Raspberry Pi 4. This object specifies the ranges the bus controller decodes on the child interface. This is analogous to the `_CRS` object, which describes the resources that the bus controller decodes on the parent interface. The ranges described in the resources of a `_DMA` object can be used by child devices for DMA or bus master transactions.

The `_DMA` object is only valid if a `_CRS` object is defined. The OSPM must reevaluate the `_DMA` object after an `_SRS` object has been executed because the `_DMA` ranges resources may change depending on how the bridge has been configured.

The following code shows the ACPI XCH0 USB 3.0 controller implementation:

```
Device (XHC0)
{
    Name (_HID, "PNP0D10")  // Hardware ID
    Name (_UID, 0x0)        // Unique ID
    Name (_CCA, 0x0)        // Cache Coherency Attribute
    Method (_CRS, 0, Serialized) {  // Current Resource Settings
        Name (RBUF, ResoureTemplate() {
            QWordMemory (ResourceConsumer,
                ,
                MinFixed,
                MaxFixed,
                NonCacheable,
                ReadWrite,
                0x0,
                SANITIZED_PCIE_CPU_MMIO_WINDOW, // MIN
                SANITIZED_PCIE_CPU_MMIO_WINDOW, // MAX
                0x0,
                0x1,                            // LEN
                ,
                ,
                MMIO
                )
            Interrupt (ResourceConsumer, Level, ActiveHigh, Exclusive, ,, ) {
                175
            }
        })
        CreateQwordField (RBUF, MMIO._MAX, MMBE)
        CreateQwoedField (RBUF, MMIO._LEN, MMLE)
        Add (MMBE, XHCI_REG_LENGTH - 1, MMBE)
        Add (MMLE, XHCI_REG_LENGTH - 1, MMLE)
        Return (RBUF)
    }

Method (_INI, 0, Serialized) {
    OperationRegion (PCFG, SystemMemory, SANITIZED_PCIE_REG_BASE + PCIE_EXT_FG_DATA,
 0x10000)
    Field (PCFG, AnyAcc, NoLock, Preserve) {
        VNID, 16, // Vendor ID
        DVID, 16, // Device ID
        CMND, 16, // Command register
        STAT, 16, // Status register
    }
    Debug = "xHCI enable"
    Store (0x6, CMND)
}
}
```

10.4 Example: UART

The system can present Arm SBSA Generic UART and 16550 UART devices. You can describe the devices with Serial Console Redirection (SPCR).

The Raspberry Pi 4 has a PL011 UART port described in `spcr.aslc` using C. The following code snippet shows the ACPI UART PL011 implementation:

```
STATIC EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE Spcr = {
  ACPI_HEADER (
    EFI_ACPI_6_3_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_SIGNATURE,
    EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE,
    EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_REVISION
  ),
  // UINT8                                   InterfaceType;
  RPI_UART_INTERFACE_TYPE,
  // UINT8                                   Reserved1[3];
  {
    EFI_ACPI_RESERVED_BYTE,
    EFI_ACPI_RESERVED_BYTE,
    EFI_ACPI_RESERVED_BYTE
  },
  // EFI_ACPI_6_3_GENERIC_ADDRESS_STRUCTURE  BaseAddress;
  ARM_GAS32 (RPI_UART_BASE_ADDRESS),
  // UINT8                                   InterruptType;
  EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_INTERRUPT_TYPE_GIC,
  // UINT8                                   Irq;
  0,                                         // Not used on ARM
  // UINT32                                  GlobalSystemInterrupt;
  RPI_UART_INTERRUPT,
  // UINT8                                   BaudRate;
#if (FixedPcdGet64 (PcdUartDefaultBaudRate) == 9600)
  EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_BAUD_RATE_9600,
#elif (FixedPcdGet64 (PcdUartDefaultBaudRate) == 19200)
  EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_BAUD_RATE_19200,
#elif (FixedPcdGet64 (PcdUartDefaultBaudRate) == 57600)
  EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_BAUD_RATE_57600,
#elif (FixedPcdGet64 (PcdUartDefaultBaudRate) == 115200)
  EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_BAUD_RATE_115200,
#else
#error Unsupported SPCR Baud Rate
#endif
  // UINT8                                   Parity;
  EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_PARITY_NO_PARITY,
  // UINT8                                   StopBits;
  EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_STOP_BITS_1,
  // UINT8                                   FlowControl;
  RPI_UART_FLOW_CONTROL_NONE,
  // UINT8                                   TerminalType;
  EFI_ACPI_SERIAL_PORT_CONSOLE_REDIRECTION_TABLE_TERMINAL_TYPE_VT_UTF8,
  // UINT8                                   Reserved2;
  EFI_ACPI_RESERVED_BYTE,
  // UINT16                                  PciDeviceId;
  0xFFFF,
  // UINT16                                  PciVendorId;
  0xFFFF,
  // UINT8                                   PciBusNumber;
  0x00,
  // UINT8                                   PciDeviceNumber;
  0x00,
  // UINT8                                   PciFunctionNumber;
  0x00,
  // UINT32                                  PciFlags;
  0x00000000,
  // UINT8                                   PciSegment;
  0x00,
```

```
      // UINT32                                    Reserved3;
      EFI_ACPI_RESERVED_DWORD
    };
```

10.5 Example: Debug port

For some OSes, the debug port is presented on the platform. To describe the debug ports available on the platform, Debug Port Table 2 is used. The table contains information about the configuration of the debug port.

The Raspberry Pi 4 has a PL011 UART port that can be described to the OS as a debug port. The following code shows the ACPI UART PL011 debug port implementation:

```
    #define RPI_DBG2_NUM_DEBUG_PORTS                        1
    #define RPI_DBG2_NUMBER_OF_GENERIC_ADDRESS_REGISTERS    1
    #define RPI_DBG2_NAMESPACESTRING_FIELD_SIZE             15

    #define RPI_UART_INTERFACE_TYPE
 EFI_ACPI_DBG2_PORT_SUBTYPE_SERIAL_ARM_PL011_UART
    #define RPI_UART_BASE_ADDRESS                 BCM2836_PL011_UART_BASE_ADDRESS
    #define RPI_UART_LENGTH                       BCM2836_PL011_UART_LENGTH
    #define RPI_UART_STR                          { '\\', '_', 'S', 'B', '.', 'G',
 'D', 'V', '0', '.', 'U', 'R', 'T', '0', 0x00 }
    STATIC DBG2_TABLE Dbg2 = {
      {
        ACPI_HEADER (
          EFI_ACPI_6_3_DEBUG_PORT_2_TABLE_SIGNATURE,
          DBG2_TABLE,
          EFI_ACPI_DBG2_DEBUG_DEVICE_INFORMATION_STRUCT_REVISION
        ),
        OFFSET_OF (DBG2_TABLE, Dbg2DeviceInfo),
        RPI_DBG2_NUM_DEBUG_PORTS
      },
      {
        /*
         * Kernel Debug Port
         */
        DBG2_DEBUG_PORT_DDI (
          RPI_DBG2_NUMBER_OF_GENERIC_ADDRESS_REGISTERS,
          RPI_UART_INTERFACE_TYPE,
          RPI_UART_BASE_ADDRESS,
          RPI_UART_LENGTH,
          RPI_UART_STR
        ),
      }
    };
```

BBR requires platforms to keep a debug port on a separate UART port from the console port so there is no conflict in debug messages and OS console output. Because the Raspberry Pi has only one active UART, enable or disable DBG2 as needed for debugging.

10.6 Example: Power button

If you remove the power cable from the device without shutting down the OS, the file system can be corrupted and other unrecoverable errors can occur. A power button is a useful addition to the embedded platform, which allows an OS to implement shutdown safely.

If we connect a button to one of the Raspberry Pi 4 GPIO pins, we can define an ACPI power button. The GPIO interrupt functionality in the BCM2711 is used with a Generic Event Device to generate the Notify command to tell OSPM that the button has been pressed. The OS then initiates sleep or soft shutdown based on user settings.

The following table shows the Generic Event Device objects:

Table 10-7: Generic Event Device Objects

Object	Description
`GED1`	Generic Event Device Object
`_HID`	Plug and Play ID: ACPI0013 for GED
`_CRS`	List of interrupts

The following table lists the Generic Event Device methods:

Table 10-8: Generic Event Device Methods

Method	Description
`_EVT`	Interrupt handler. This has arg0, which contains the Generic System Interrupt Vector of the interrupt.
`_INI`	Platform Specific Initialization

The following table shows the power button objects:

Table 10-9: Power Button Objects

Object	Description
PWRB	Power Button object
`_HID`	Plug and Play ID: PNP0C0C for power button

The following table lists the power button methods:

Table 10-10: Power Button Methods

Method	Description
`_STA`	Status of the device. We return `0xF`, which means the device is present, enabled, should be shown in UI and is functioning properly.

Using the `_INI` method, set up GPIO pin 5 to trigger an interrupt when a rising edge is detected. Then, in the `_EVT` method, check the status of the pins to check that the interrupt was GPIO0, and that pin 5 triggered the interrupt. If the interrupt is triggered, the status is reset and the power button notified.

The following code shows an ACPI power button implementation:

```
// Generic Event Device
Device (GED1) {
  Name (_HID, "ACPI0013")
  Name (_UID, 0)


  Name (_CRS, ResourceTemplate () {
    Interrupt(ResourceConsumer, Edge, ActiveHigh, ExclusiveAndWake) {
    BCM2386_GPIO_INTERRUPT0 }
  })


  OperationRegion (PH0, SystemMemory, GPIO_BASE_ADDRESS, 0x1000)
  Field (PH0, DWordAcc, NoLock, Preserve) {
    GPF0, 32, /* GPFSEL0 - GPIO Function Select 0 */
    offset(0x40),
    GPE0, 32, /* GPEDS0 - GPIO Pin Event Detect Status 0 */
    GPE1, 32, /* GPEDS1 - GPIO Pin Event Detect Status 1 */
    GRE0, 32, /* GPREN0 - GPIO Pin Rising Edge Detect Enable 0 */
    GRE1, 32, /* GPREN1 - GPIO Pin Rising Edge Detect Enable 1 */
    offset(0xe4),
    GUD0, 32, /* GPIO_PUP_PDN_CNTRL_REG0 - GPIO Pull-up / Pull-down
 Register 0 */
  }


  Method (_INI, 0, NotSerialized) {
    /* 0x00000020 = GPIO pin 5 */
    /* Enable rising edge detect */
    Store(0x00000020, GRE0)
    /* Enable Pull down resistor for pin 5 */
    Store(0x00000800, GUD0)
  }


  Method (_EVT, 1) {
    If (ToInteger(Arg0) == BCM2386_GPIO_INTERRUPT0)
      Name()
      Store(0x00000020, GPE0) // Clear the status
      Notify (\_SB.PWRB, 0x80) // Sleep/Off Request
    }
  }
}


Device (PWRB) {
  Name (_HID, "PNP0C0C")
  Name (_UID, Zero)
  Method (_STA, 0x0, NotSerialized) {
    Return(0xF)
  }
}
```

10.7 Example: PCIe ECAM

If a system has a BSA-compliant PCIe controller, the platform can simply report PCIe Configuration Space using the ACPI MCFG table to achieve the best OS compatibility. PCIe then just works on generic off-the-shelf OSes.

However, if a system's PCIe controller is not BSA-compliant, you can still improve OS compatibility by implementing either of the following two alternatives below:

1. Expose a PCIe device attached to the non-compliant PCI controller in the DSDT table if the device is supported in ACPI such as the USB XHCI controller.
2. Alternatively, you can use the Arm PCI Configuration Space Access Firmware Interface. You can use this interface as an alternative to the Enhanced Configuration Access Mechanism (ECAM) hardware mechanism.

The Arm PCI Configuration Space Access Firmware enables a caller to:

- Access PCI configuration space reads and writes
- Discover the implemented PCI segment groups and bus ranges for each segment

For the list of supported calls, see the Arm PCI Configuration Space Access Firmware Interface.

Arm PCI Configuration Space Access Firmware Interface implementation requires the following:

- On the platform with EL3 presented, Platform Firmware SMCCCv1.1 compliant implementation
- If EL3 is not present but EL2 is present, HVC conduit must be implemented in hypervisor
- Operating System SMCCv1.1 compliant SMC or HVC conduit implementation

Enabling Arm PCI Configuration Space Access Firmware Interface requires patches for a platform firmware, UEFI, and an OS.

An example of the SMCCC implementation supporting Arm PCI Configuration Space Access Firmware Interface is in Arm Trusted Firmware. Arm Trusted Firmware allows platforms to handle PCI configuration access requests through standard SMCCC. To enable these access requests, the SMC_PCI_SUPPORT build flag is provided.

To use PCIe SMCCC, describe PCIe Root Complex in the SSDT ACPI table. Refer to this patch [PATCH v2 3/6] Platform/RaspberryPi: Add PCIe SSDT. With this patch, instead of hiding the PCIe root complex, expose PCIe to the OS. The OS ACPI PCI driver controls the PCIe root complex but because the MCFG table is absent, the driver uses the OS SMC conduit to get access to the PCIe ECAM.

An example of the OS SMC conduit implementation is in the NetBSD. NetBSD implements pci_smccc_call(), which uses a Secure Monitor Call to request a PCI Configurate access service to a platform firmware running in EL3. With PCI_SMCCC enabled, the NetBSD PCIe subsystem uses the PCI_VERSION SMC call to check if the SMCCC supports PCI configuration access. If the SMCCC version is 1.1 or later, the PCI SMCCC is supported.

You can build and run NetBSD, Arm Trusted Firmware and EDK2 on the Raspberry Pi 4 with PCI SMCCC enabled. As a result, the PCIe is exposed through SMCCC driving the XHCI controller.

In the future, other operation systems or hypervisors such as VMWare ESXi might implement this interface.

10.8 ACPI integration recommendations

You can implement ACPI tables using a platform driver or dynamic ACPI framework.

For platform drivers, you manually create ACPI tables using ACPI Source Language (ASL). Create a set of `.asl` files and an edk2 module information file `AcpiTable.inf`. You can also create an ACPI table using C language. In this case, `.aslc` files must be used.

These files are compiled at build time and stored in a firmware volume. At boot time, a platform driver uses ArmLib methods, shown in the following code:

```
EFI_STATUS LocateAndInstallAcpiFromFvConditional(
 IN CONST EFI_GUID* AcpiFile,
 IN EFI_LOCATE_ACPI_CHECK CheckAcpiTableFunction
)
or
EFI_STATUS LocateAndInstallAcpiFromFv(
 IN CONST EFI_GUID* AcpiFile
)
```

These methods locate and install ACPI tables in a firmware volume. The following code snippet locates ACPI tables implemented for the platform and installs it in a firmware volume:

```
Status = LocateAndInstallAcpiFromFv(&mAcpiTableFile);
```

In this example, `mAcpiTableFile` is a GUID of the ACPI storage file in a firmware volume and matches `FILE_GUID` in the `AcpiTable.inf`.

Although ACPI tables are compiled at build time and stored in a firmware volume, you can modify these tables at boot time. The second parameter `CheckAcpiTableFunction` in `LocateAndInstallAcpiFromFvConditional()` is a pointer to a function. This parameter is an algorithm `LocateAndInstallAcpiFromFvConditional()` used to locate and install ACPI, and performs the following steps:

1. Use `EFI_FIRMWARE_VOLUME2_PROTOCOL` and `mAcpiTableFile` GUID to find an ACPI table in a firmware volume.
2. Prior to the installation of the table, call `CheckAcpiTableFunction()` with a pointer to a newly found ACPI table as a parameter.
3. Provided `CheckAcpiTableFunction()` indicates that the table should be installed, use EFI_ACPI_TABLE_PROTOCOL to install the table.
4. Repeat until all ACPI tables are found and installed.

`CheckAcpiTableFunction()` has a pointer to a newly discovered ACPI table and can modify the table before being installed. For an example, see the `HandleDynamicNamespace()` function of the Raspberry Pi 4 ACPI platform driver and see how it is used to modify DSDT and SSDT ACPI tables with values taken from PCD values.

For a Raspberry Pi 4 ACPI table implementation, see AcpiTables.

To learn how ACPI tables are installed on the Raspberry Pi 4, see ConfigDxe.

For another example of the ACPI platform driver, see PlatformDxe. The dynamic ACPI table generators that are implemented as libraries. These generators query a platform-specific Configuration Manager to collate the information required for generating the tables at runtime. See Arm at master for a list of the generators supported.

To implement Configuration Manager, include a platform-specific DXE driver called ConfigurationManagerDxe. Configuration Manager produces EDKII_CONFIGURATION_MANAGER_PROTOCOL and implements its API. The declaration of the API for the EDKII_CONFIGURATION_MANAGER_PROTOCOL is in ConfigurationManagerProtocol.h.

The following code shows the GUID of the Configuration Manager Protocol:

```
    #define EDKII_CONFIGURATION_MANAGER_PROTOCOL_GUID       \
      { 0xd85a4835, 0x5a82, 0x4894,                         \
        { 0xac, 0x2, 0x70, 0x6f, 0x43, 0xd5, 0x97, 0x8e }   \
      };
```

The following code shows a software interface of the Configuration Manager Protocol:

```
typedef struct ConfigurationManagerProtocol {
  UINT32                                Revision;
  EDKII_CONFIGURATION_MANAGER_GET_OBJECT  GetObject;
  EDKII_CONFIGURATION_MANAGER_SET_OBJECT  SetObject;
  EDKII_PLATFORM_REPOSITORY_INFO        * PlatRepoInfo;
} EDKII_CONFIGURATION_MANAGER_PROTOCOL;
```

The API consists of the following functions:

- `GetObject()`. The GetObject() function defines the interface implemented by the Configuration Manager Protocol used to return the Configuration Manager Objects
- `SetObject()`. The SetObject() function defines the interface implemented by the Configuration Manager Protocol to update the Configuration Manager Objects

Configuration Manager Objects are objects that represent platform configuration and are stored in the EDKII_PLATFORM_REPOSITORY_INFO repository, maintained by Configuration Manager.

Configuration Manager maintains a list of ACPI tables to be installed. Based on this list, the corresponding ACPI table generators are invoked by the Dynamic ACPI framework.

For example, the IORT ACPI table generator handles the following ACPI objects:

- EArmObjItsGroup
- EArmObjNamedComponent
- EArmObjRootComplex
- EArmObjSmmuV1SmmuV2
- EArmObjSmmuV3
- EArmObjPmcg
- EArmObjGicItsIdentifierArray
- EArmObjIdMappingArray
- EArmObjGicItsIdentifierArray

If the OEM platform has an SMMUv3 hardware block, include an object with ID equal to EArmObjSmmuV3 in the Configuration Manager repository. For more information, refer to the list of Arm object IDs and data structures in ArmNameSpaceObjects.h.

The IORT ACPI table generator requests the EArmObjSmmuV3 object using the EDKII_CONFIGURATION_MANAGER_GET_OBJECT function and adds the SMMUv3 node to the IORT ACPI table. The same mechanism is used by other ACPI table generators.

For an implementation example, see ConfigurationManager for EDKII_CONFIGURATION_MANAGER_PROTOCOL.

Note

Currently, the capability to generate ASL tables (DSDT and SSDT) is limited to generating ASL Serial Port Information corresponding to DBG2 and SPCR because it is platform-specific.

---

11. SMBIOS requirements and implementations

The SMBIOS table version 3.0.0 or later is required to conform to the SMBIOS specification. Earlier SMBIOS table and format versions are not supported.

For information about the required SMBIOS records for SystemReady band compliant systems, see the Arm Base Boot Requirement (BBR) specification. For example, the Raspberry Pi 4 uses the following SMBIOS records:

- Type 00: BIOS information
- Type 01: system information
- Type 02: base board information (optional)
- Type 03: chassis information
- Type 04: processor information
- Type 07: cache information
- Type 09: system slot information
- Type 11: OEM string (optional)
- Type 16: physical memory array
- Type 17: memory device
- Type 19: memory array mapped address
- Type 32: boot status

11.1 SMBIOS integration

SMBIOS data structures are built on top of the platform-independent driver SmbiosDxe, which uses the EFI_SMBIOS_PROTOCOL API. EFI_SMBIOS_PROTOCOL allows consumers to log SMBIOS data records and enables the producer (SmbiosDxe) to create the SMBIOS tables for a platform. SmbiosDxe is responsible for installing the pointer to the tables in the EFI System Configuration Table.

The following code shows a GUID of SMBIOS Protocol:

```
    #define EFI_SMBIOS_PROTOCOL_GUID \
    { 0x3583ff6, 0xcb36, 0x4940, { 0x94, 0x7e, 0xb9, 0xb3, 0x9f,\
    0x4a, 0xfa, 0xf7 } }
```

The following code shows an SMBIOS Protocol data structure:

```
typedef struct _EFI_SMBIOS_PROTOCOL {
 EFI_SMBIOS_ADD Add;
```

```
 EFI_SMBIOS_UPDATE_STRING UpdateString;
 EFI_SMBIOS_REMOVE Remove;
 EFI_SMBIOS_GET_NEXT GetNext;
 UINT8 MajorVersion;
 UINT8 MinorVersion;
} EFI_SMBIOS_PROTOCOL;
```

11.2 Platform driver

The SMBIOS driver is a platform-specific DXE driver that uses SMBIOS data records provided by the OEM. The driver consumes EFI_SMBIOS_PROTOCOL, which is produced by SmbiosDxe and uses its interface to add SMBIOS records.

The driver creates SMBIOS records defined in SmBios.h. These records are standard SMBIOS data structures, defined according to the latest SMBIOS specification.

For example, the following code shows the definition for a TYPE 1 System information SMBIOS table, which is defined by the PlatformSmbiosDxe Raspberry Pi 4 platform driver:

```
SMBIOS_TABLE_TYPE1 mSysInfoType1 = {
  { EFI_SMBIOS_TYPE_SYSTEM_INFORMATION, sizeof (SMBIOS_TABLE_TYPE1), 0 },
  1,    // Manufacturer String
  2,    // ProductName String
  3,    // Version String
  4,    // SerialNumber String
  { 0x25EF0280, 0xEC82, 0x42B0, { 0x8F, 0xB6, 0x10, 0xAD, 0xCC, 0xC6, 0x7C,
 0x02 } },
  SystemWakeupTypePowerSwitch,
  5,    // SKUNumber String
  6,    // Family String
};
```

PlatformSmbiosDxe uses EFI_SMBIOS_PROTOCOL method `Add()` to add mSysInfoType1 record:

```
Status = gBS->LocateProtocol (&gEfiSmbiosProtocolGuid, NULL, (VOID**)&Smbios);
Status = Smbios->Add (
                    Smbios,
                    gImageHandle,
                    &c,
                    Record // mSysInfoType1
                  );
```

The platform driver is responsible for ensuring that the SMBIOS record is formatted to match the version of the SMBIOS specification as defined in the MajorVersion and MinorVersion fields of the EFI_SMBIOS_PROTOCOL.

Add both a platform driver and SmbiosDxe driver to your platform and flash description files. Use the RPi4.dsc and the RPi4.fdf files as a reference.

For more information about how the platform driver is implemented on the Raspberry Pi 4, see the PlatformSmbiosDxe implementation.

11.3 System Management BIOS framework

The platform driver requires the OEM to define SMB records using C and check that these records are formatted according to the version of the SMBIOS specification as defined in the MajorVersion and MinorVersion fields of the EFI_SMBIOS_PROTOCOL.

The generic Arm System Management BIOS (SMBIOS) framework allows you to generate SMBIOS tables without writing C code. This framework uses platform configuration PCD database entries and strings from a Human Interface Infrastructure (HII).

For example, the OEM can provide the following PCD entries in its platform description file:

- `gEfiMdeModulePkgTokenSpaceGuid.PcdFirmwareVendor`
- `gEfiMdeModulePkgTokenSpaceGuid.PcdFirmwareVersionString`
- `gArmTokenSpaceGuid.PcdSystemBiosRelease`
- `gArmTokenSpaceGuid.PcdEmbeddedControllerFirmwareRelease`

These entries are taken by the SMBIOS framework and added to the SMBIOS table type 00 BIOS information automatically.

The OEM must provide an OemMiscLib library with the following platform-specific definitions:

Processor Information

The SMBIOS framework creates processor and cache information tables and requires the following functions:

- `OemGetCpuFreq()`
- `OemGetProcessorInformation()`
- `OemGetCacheInformation()`
- `OemGetMaxProcessors()`

The SMBIOS framework calls these functions to get processor and cache information and uses the EFI_SMBIOS_PROTOCOL `Add()` function to add SMBIOS type 04 and type 07 tables.

OemUpdateSmbiosInfo() function

The SMBIOS framework uses hardcoded PCD entries to create SMBIOS tables, but platform-specific information is needed in runtime. For example, a baseboard serial number or chassis serial number must not be hardcoded in the UEFI binary the OEM uses to flash the board. The OEM can write `OemUpdateSmbiosInfo()` so that these two strings are read in runtime from a baseboard management controller. The SMBIOS framework calls `OemUpdateSmbiosInfo()` to retrieve these two strings and update default information in the SMBIOS type 02 and type 03 tables.

For more details about the OemMiscLib implementation, see `tianocore/edk2-platforms/Platform`.

For more information about the SMBIOS framework, see https://github.com/tianocore/edk2/tree/master/ArmPkg/Universal/Smbios/SmbiosMiscDxe.

12. UEFI requirements

The boot and system firmware for 64-bit Arm embedded servers is based on the UEFI specification version 2.8 or later and incorporates the AArch64 bindings.

UEFI compliant systems must follow the requirements in section 2.6 of the specification. However, to ensure a common boot architecture for server-class AArch64, systems compliant with this specification must provide the UEFI services and protocol from the provided list.

UEFI compliance is tested using UEFI Self-Certification Tests (SCT) and FWTS. For more information about using SCT and FWTS, see ACS.

For a list of required UEFI runtime and boot services, see the Arm Base Boot Requirements specification.

13. Related information

The following resources are related to material in this guide.

Specifications:

- Arm Base System Architecture (BSA) specification
- Arm Server Base System Architecture (SBSA) specification
- Arm Base Boot Requirements (BBR) specification
- Base Boot Security Requirements (BBSR)
- Arm SystemReady Requirements Specification

Repositories:

- Arm SystemReady ACS Repository

User Guides:

- SystemReady Band Policy Guidelines
- SystemReady FAQ

SystemReady Pages:

- Arm SystemReady Band Page
- Arm SystemReady Compliance Program
- Arm Community - SystemReady Forum

Other Resources:

- UEFI Self-Certification Test
- Advanced Configuration and Power Interface (ACPI) Specification
- Arm PCI Configuration Space Access Firmware Interface

Appendix A Running ACS tests manually

To run ACS tests manually, select the **UEFI Execution Environment** option from the ACS grub menu to boot UEFI shell. Then, navigate to the `acs_tests` folder on the ACS drive partition.

Figure A-1: acs_tests folder contents on page 73 shows the folder contents:

Figure A-1: acs_tests folder contents

```
FS0:\> cd acs_tests
FS0:\acs_tests\> ls
Directory of: FS0:\acs_tests\
11/21/2024  13:45 <DIR>         4,096  .
11/21/2024  13:45 <DIR>             0  ..
11/21/2024  13:45 <DIR>         4,096  app
06/16/2021  16:35 <DIR>         4,096  bbr
11/21/2024  13:45 <DIR>         4,096  bbsr-keys
11/21/2024  13:45 <DIR>         4,096  bsa
11/21/2024  13:45 <DIR>         4,096  config
11/21/2024  13:45 <DIR>         4,096  debug
11/21/2024  13:45 <DIR>         4,096  parser
          0 File(s)           0 bytes
          9 Dir(s)
FS0:\acs tests\>
```

In this directory, the `bbr` folder contains the UEFI Self-Certification Test and the bsa folder has a UEFI shell application for BSA and SBSA compliance. For more information, see sysarch-acs.

To run the bsa and sbsa tests, go to the folder and start the application using the following command:

```
FS0:\acs_tests\bsa\> bsa.efi
```

or

```
FS0:\acs_tests\bsa\sbsa\> sbsa.efi
```

For a list of application parameters, refer to the Compliance User Guides available at SYSARCH-ACS Docs.

To run the same SCT command as the automated process, that is to run all the SCT SBBR tests, go to the folder and start the application using the following command:

```
FS0:\acs_tests\bbr\sct\> SCT.efi -s SBBR.seq
```

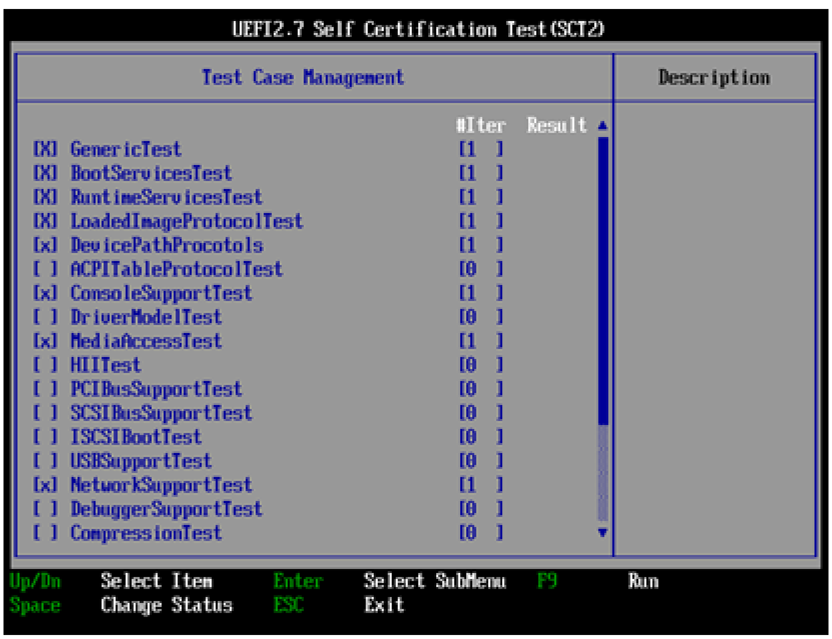
Figure A-2: Start SCT with a GUI on page 74 shows how to run specific SCT test cases, you can start SCT with a GUI by passing `-u` as a parameter:

Figure A-2: Start SCT with a GUI

```
FS0:\acs_tests\bbr\sct\> ls
Directory of: FS0:\acs_tests\bbr\sct\
11/21/2024  13:45 <DIR>          4,096  .
11/21/2024  13:45 <DIR>          4,096  ..
11/21/2024  13:45 <DIR>          4,096  Application
06/16/2021  16:35 <DIR>          4,096  Data
11/21/2024  13:45 <DIR>          4,096  Dependency
11/21/2024  13:45 <DIR>          4,096  Ents
06/16/2021  16:35 <DIR>          4,096  Log
06/16/2021  16:35 <DIR>          4,096  Overall
11/21/2024  13:45 <DIR>          4,096  Proxy
06/16/2021  16:36 <DIR>          4,096  Report
11/21/2024  13:45 <DIR>          4,096  SCRT
11/21/2024  13:45              243,200  SCT.efi
06/16/2021  16:35                  150  Sct.log
11/21/2024  13:45 <DIR>          4,096  Sequence
11/21/2024  13:45               26,112  StallForKey.efi
11/21/2024  13:45 <DIR>          4,096  Support
11/21/2024  13:45 <DIR>         12,288  Test
          3 File(s)     269,462 bytes
         14 Dir(s)
FS0:\acs_tests\bbr\sct\> sct -u
```

1. Press F5 to select tests manually. Press Enter.
2. Figure A-3: Test Case Management menu on page 74 shows the view, add, or remove tests in the Test Case Management menu:

Figure A-3: Test Case Management menu



3. Press F9 to run SCT, as shown in Figure A-4: SCT screen on page 75:

Figure A-4: SCT screen

Note

Sometimes SCT can hang in the process of self-reset. In this case, power off the system then power it on. The tests are not reset. During the next boot if the same drive with SCT has been selected as the boot device, the test continues. Follow the steps outlined in Boot order to ensure the ACS drive is the first boot option.

If you need to build your own image, ACS tests are open source and can be downloaded from SystemReady ACS. Read the documentation in this repository to learn how to build and construct test images.

Appendix B Set up the Raspberry Pi 4

This section describes how to set up a Raspberry Pi 4 for compliance testing and development.

Note that although Raspberry Pi 4 is a SystemReady ES certified system, it is still not SystemReady band compliant. It has some compliance issues, but can still be used by people who would like to get familiar with compliance testing and development. For SystemReady band compliant systems, please refer to the sections for RD-N2 FVP and Radxa Orion O6. Both are certified with SystemReady SR with SBSA level 6 and are also SystemReady band compliant.

To set up the Raspberry Pi, you need the following hardware:

Power

A powered USB hub to avoid overloading the standard Raspberry Pi power supply.

Network controller (NIC)

UEFI supports the Raspberry Pi NIC such as for Preboot eXecution Environment (PXE) booting. However, the NIC driver is missing from many OS distributions. Use a USB NIC, such as a Realtek RTL8153 based device. For this guide, we tested the Raspberry Pi with RTL8153 NIC.

Storage

A micro SD card and a USB storage device. The micro SD holds the UEFI firmware and any FAT16 or FAT32 capable drive will work.

The USB Storage device is the main disk for the operating system. Connect it to the USB port of the Raspberry Pi. We recommend the USB 3.0 blue ports for better performance.

Check your OS for minimum install size, for example, 64 to 128GB as a starting point. You can use thumb drivers and drive enclosures. We recommend a UASP enabled external drive. A second 8GB or larger thumb drive is recommended for the OS installer.

Interfacing

Use the Raspberry Pi video output with a keyboard and mouse or use a serial connection. You can setup both types of connection at the same time.

Keyboard and mouse

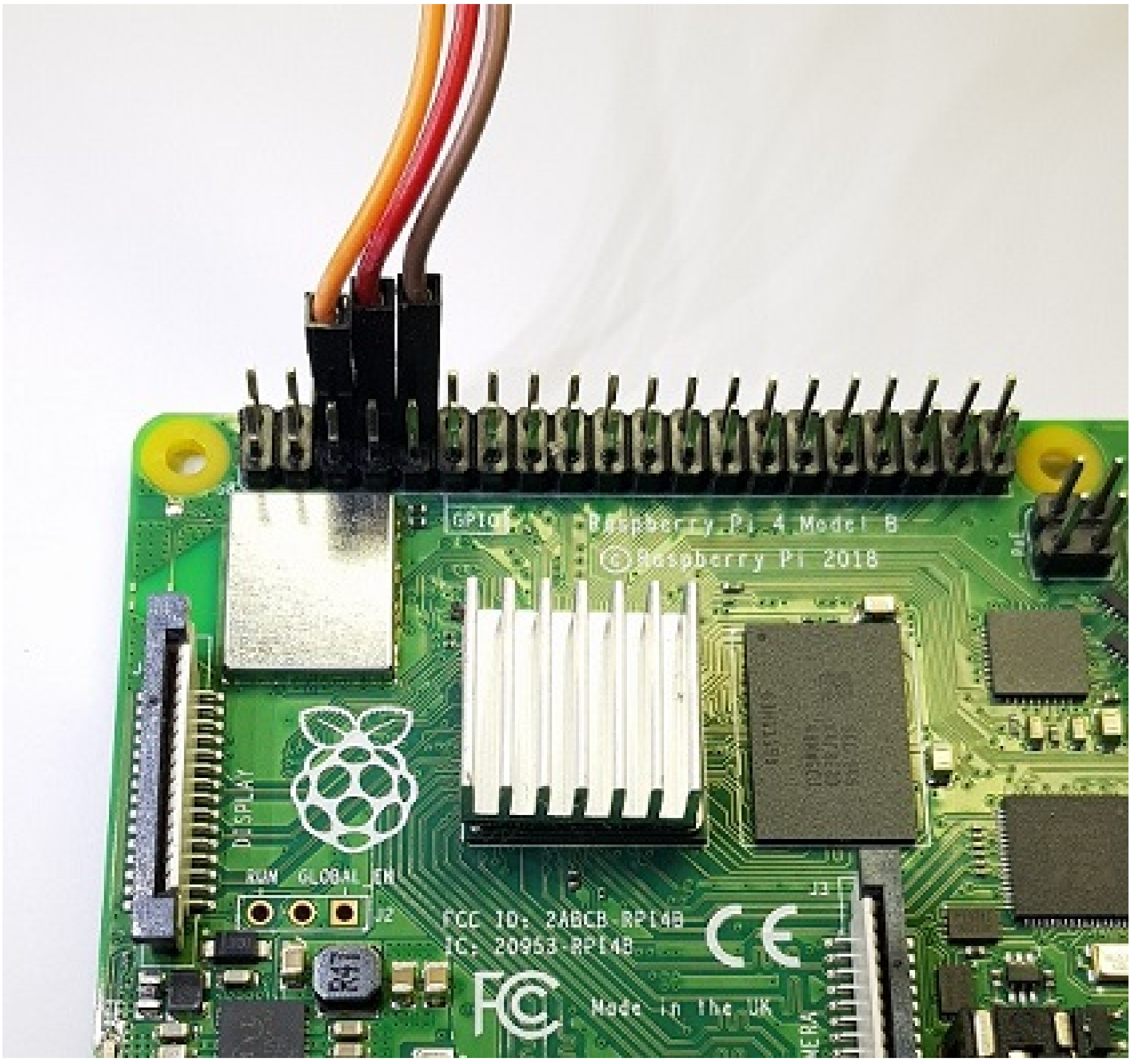
Use an HDMI micro to HDMI cable and an HDMI display to output the video. USB mice and keyboards with generic drivers will work.

Serial adapter

For this guide, use a generic TTL serial adapter that utilizes separate cables. You need to use three of the wires.

Figure B-1: Raspberry Pi serial adapter connections on page 77 shows how to connect the serial adapter to your Raspberry Pi:

Figure B-1: Raspberry Pi serial adapter connections



The following table shows the connection details:

Table B-1: Connection Details

Description	TX	RX	GRND
Color	Red	Brown	Orange
Header pin	8	10	6
GPIO	GPIO14	GPIO15	•

Finally, connect the serial cable USB connector to your PC.

B.1 Set up the terminal

If you are using Windows, you need a terminal emulator such as PuTTY.

The following table shows the configuration required, and the following text describes how to set up your connection with PuTTY:

Table B-2: PuTTy Configuration

Variable	Value
Baud rate	115200
Data bits	8
Parity	None
Stop bits	1

1. On the **Session** configuration panel in PuTTY, select **Serial** from the **Connection type** options.
2. Use the **Serial line** and **Speed** options to specify which serial line to use and the Baud rate to use to transfer data.
3. For more information on serial connection with PuTTY, see Connecting to a local serial line.

If you are using Linux or a Mac, use terminal emulators such as minicom or screen to connect to the TTL serial connection. If there are no serial devices connected to your computer, your serial connector is `/dev/ttyUSB0`. If you have more than one serial device, use a tool such as `dmseg` to check `ttyUSB<num>`.

To connect using `screen`, enter the following command:

```
$ screen /dev/ttyUSB0 115200
```

To connect using minicom, enter the following command:

```
$ minicom -D /dev/ttyUSB0
```

For more information and troubleshooting, see Using a console cable with Raspberry Pi.

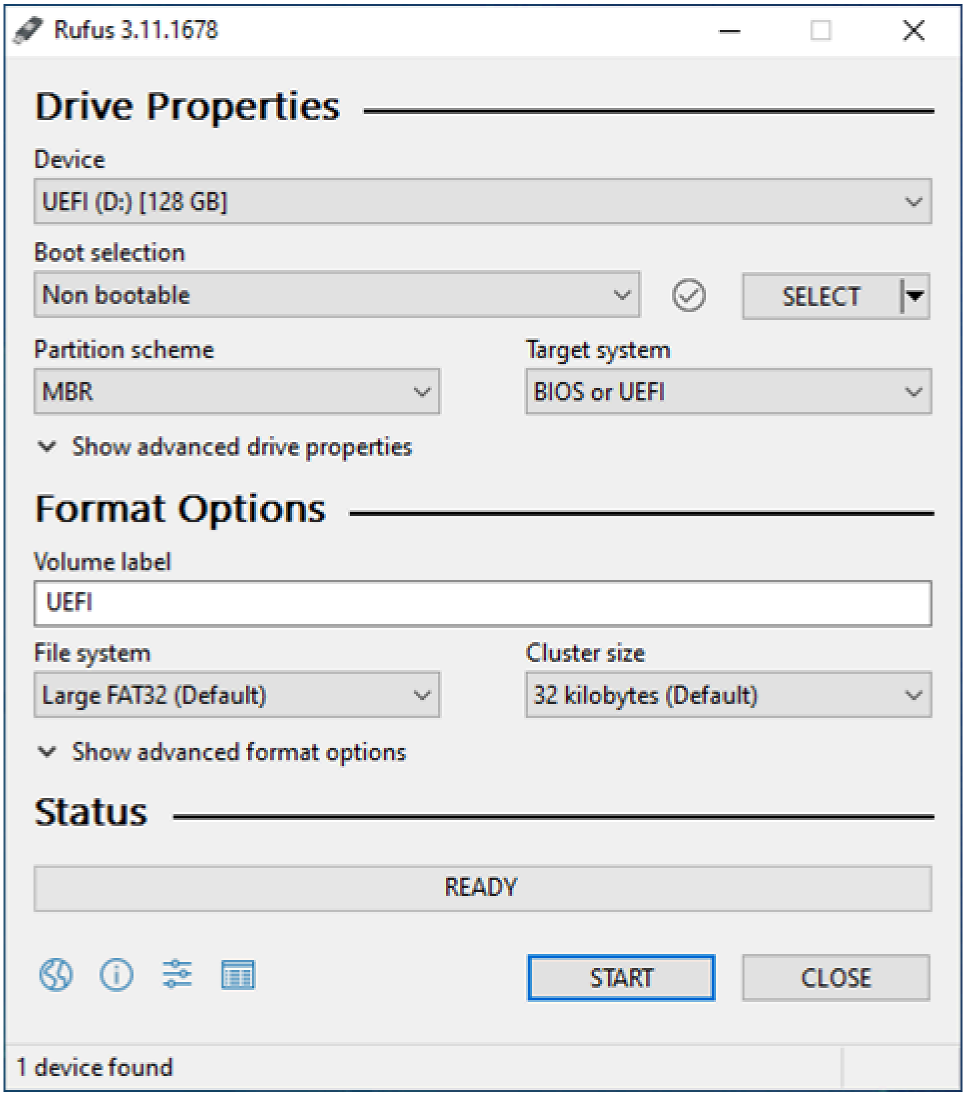
B.2 Format the SD drive

For Raspberry Pi 4, you can format the SD drive in Large FAT16 or Large FAT32 for updating the EEPROM and storing the UEFI firmware.

To format the SD drive on Windows, use Rufus and the following procedure:

1. In Rufus, select your device then select **Non bootable** from the **Boot selection** menu. Ensure the file system type is Large FAT16 or Large FAT32. Figure B-2: Rufus format options on page 79 shows the file system type:

Figure B-2: Rufus format options

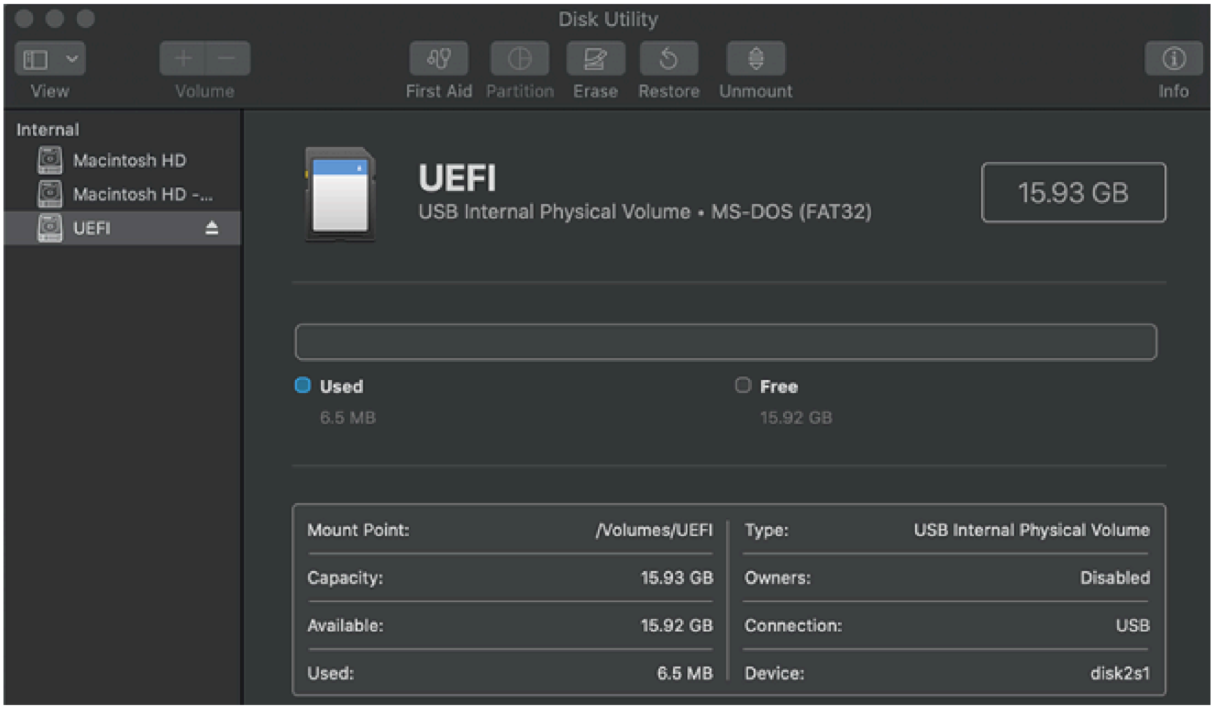


2. Click **Show advanced format options** and disable **Create extended label and icon files**. This option is not needed.
3. Click **START**.

To format the drive on Mac OS:

1. Open Disk Utility and select your SD card in the list of drives as shown in Figure B-3: Disk Utility window on Mac on page 80:

Figure B-3: Disk Utility window on Mac

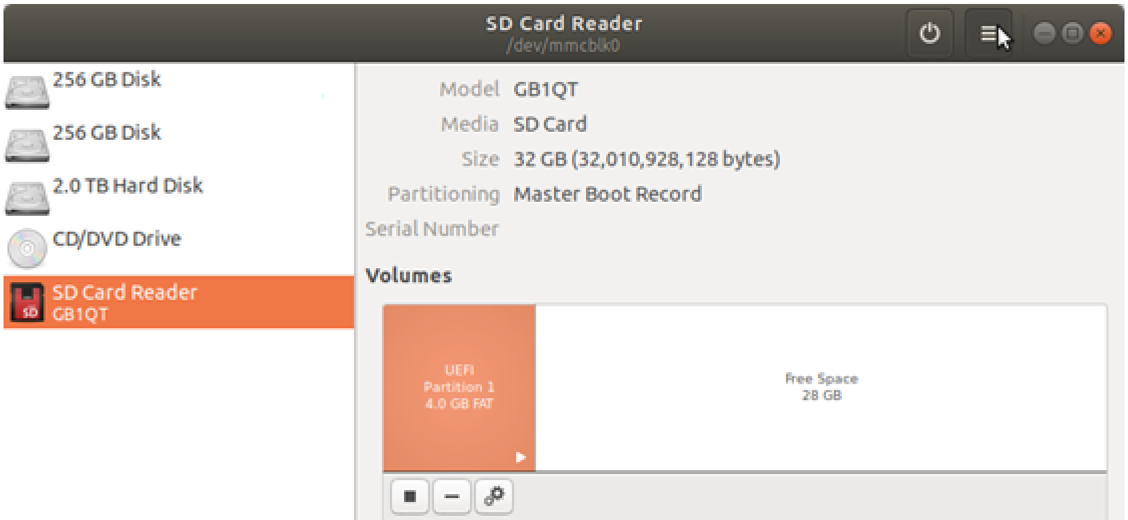


2. Click **Erase** to format the drive.
3. In the format list, select MS-DOS (FAT).

To format the drive on Linux:

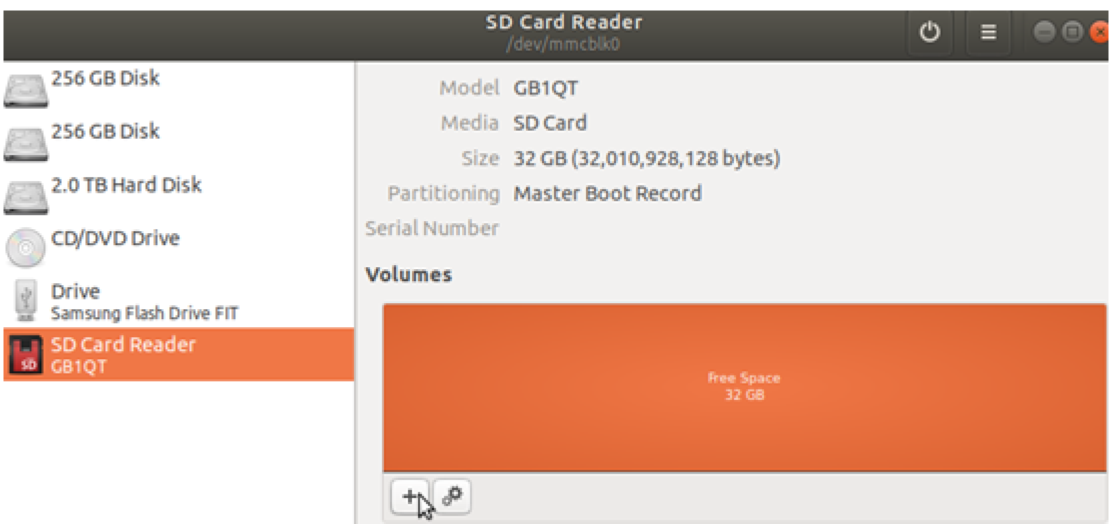
1. Use either graphical or command-line instructions. When using graphical instructions, open Disks and select your SD card.
2. Figure B-4: Disk format option on page 81 shows where to click on the bars at the top of the window.:

Figure B-4: Disk format option



3. Select **Format Disk**, then select **Compatible with all systems and devices (MBR/DOS)**.
4. Click **Format**. A blank formatted disk is created.
5. Figure B-5: Add partition on page 81 shows where to click **+** to add a partition:

Figure B-5: Add partition



6. Select a **Partition Size**. For this guide, the firmware image is under 10MB, so any partition size can be used. Click **Next**.
7. In **Type**, select **For use with all systems and devices (FAT)**. Click **Create**.

B.3 Update the EEPROM

To update the EEPROM:

1. Ensure the Raspberry Pi 4 is running the latest firmware on the EEPROM.
2. Download the latest version of `rpi-eeprom` from RPi eeprom github and use this tool to update the boot EEPROM.
3. Unzip the contents of `rpi-boot-eeprom-recovery` to a blank, FAT formatted SD-SDCARD.
4. Power off the Raspberry Pi 4.
5. Insert the SD card.
6. Power on the Raspberry Pi 4 and wait 10 seconds.

The green LED light blinks rapidly to show success. Otherwise, an error pattern is displayed.

If an HDMI display is attached to the Raspberry Pi 4, the screen shows green for success or red if a failure occurs.

B.4 Install UEFI

The latest UEFI binaries and installation guide are on PFTF Github.

To install UEFI:

1. Download the latest archive from Releases.
2. Create an SD card or a USB drive with at least one partition. This can be a regular partition or an ESP. Format the partition to FAT16 or FAT32.

Note

To boot from USB or ESP, you need the latest version of firmware on EEPROM. If you are using the latest UEFI firmware and you cannot boot from USB or ESP, see Update the EEPROM.

3. Extract all the files from the downloaded archive to the partition you created. Do not change the names of the extracted files and directories.

To run UEFI:

1. Insert the SD card or connect the USB drive and power up your Raspberry Pi 4. A multicolored screen shows the embedded bootloader reading the data. The Raspberry Pi 4 logo appears when the UEFI firmware is ready.
2. Press Esc to enter the firmware setup, F1 to launch the UEFI Shell, or wait for the UEFI boot option to boot Raspberry Pi 4.

You can build UEFI firmware from source. The following steps are for Ubuntu Linux 18.04.1 on x86_64 host PC using cross compilation.

To build UEFI firmware:

1. Create a workspace directory with the following commands:

```
$ mkdir RPi4
$ export WORKSPACE=$(pwd)/RPi4
```

2. Clone the pftf/RPi4 repository:

```
$ git clone http://github.com/pftf/RPi4.git
$ git submodule update -init
```

3. Initialize submodules for both the `edk2` and `edk2-platform` repositories using the commands shown:

```
$ cd edk2
$ git submodule update -init
$ cd ../edk2-platforms
$ git submodule update -init
$ cd ..
```

4. Copy `0001-MdeModulePkg-UefiBootManagerLib-Signal-ReadyToBoot-o.patch` to the `edk2` folder and run the following command:

```
$ patch -p3 < 0001-MdeModulePkg-UefiBootManagerLib-Signal-ReadyToBoot-o.patch
```

5. Install a toolchain for cross compilation using the following command:

```
$ sudo apt-get install gcc-aarch64-linux-gnu
```

6. Follow the instructions on Building EDKII UEFI firmware for Arm Platforms to build a binary. An example of the build command for RPi4 platform follows:

```
$ GCC5_AARCH64_PREFIX=aarch64-linux-gnu-
$ build -n 8 -a AARCH64 -t GCC5 -p Platform/RaspberryPi/RPi4/RPi4.dsc
```

   The resulting binary `RPI_EFI.fd` is found in the `RPi4/Build/<BUILD_TARGET>/FV` folder.

7. Follow the steps in the "Booting the firmware section" in Raspberry Pi 4 Platform to prepare a bootable SD card.

B.5 Configure UEFI

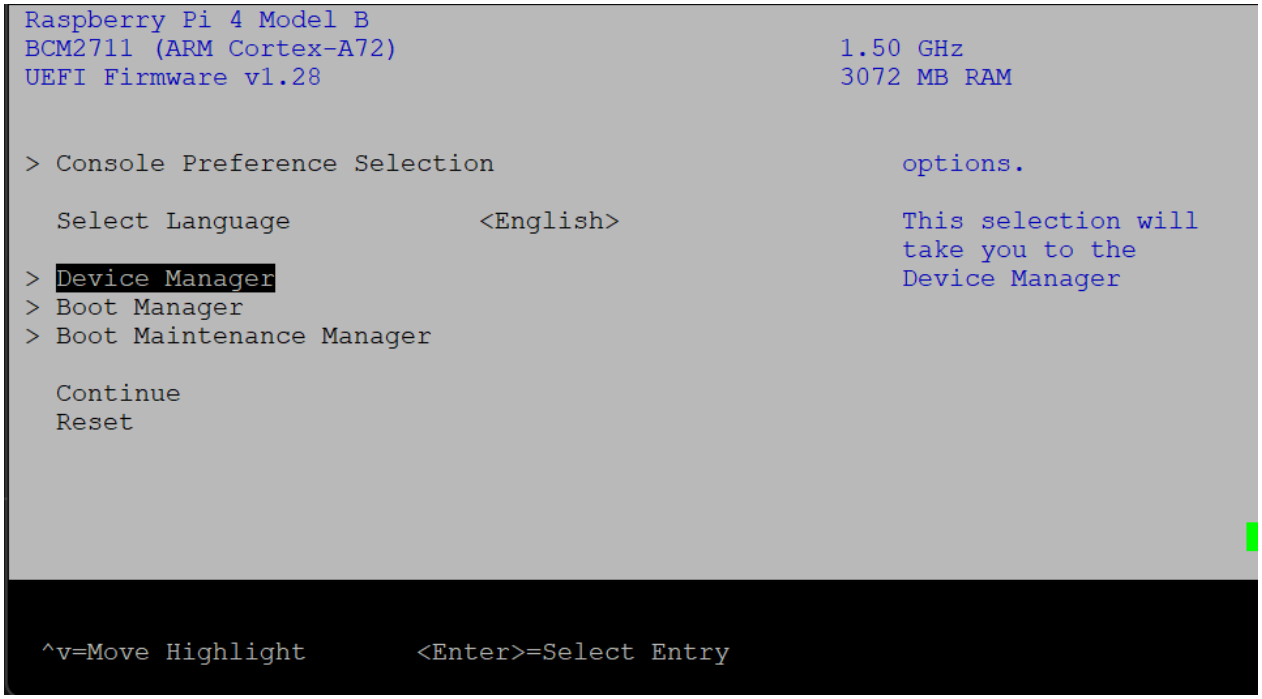
Figure B-6: UEFI shell screen on page 84 shows the boot into the UEFI shell by pressing F1 during the boot process:

Figure B-6: UEFI shell screen

```
UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (https://github.com/pftf/RPi4, 0x00010000)
Mapping table
      FS1: Alias(s):HD1b:;BLK3:
          VenHw(100C2CFA-B586-4198-9B4C-1683D195B1DA)/HD(1,MBR,0x0E26C215,0x800,
0x3B71800)
      FS0: Alias(s):HD0c0b:;BLK1:
          PcieRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/USB(0x2,0x0)/HD(1,GPT,A39411BF
-9A10-41FA-9FAD-3A2303FAF354,0x800,0x13FFFF)
     BLK2: Alias(s):
          VenHw(100C2CFA-B586-4198-9B4C-1683D195B1DA)
     BLK0: Alias(s):
          PcieRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/USB(0x2,0x0)
Press ESC in 2 seconds to skip startup.nsh or any other key to continue.
Shell>
```

To boot to the UEFI menu, press Esc during the boot process. The following UEFI menu is displayed:

Figure B-7: UEFI menu



In this menu, you can change device settings and manually boot the device using Boot Manager.

B.6 Troubleshooting UEFI

To boot to the UEFI menu:

1. Press Esc to interrupt the boot process.
2. In the UEFI menu, navigate to the Boot Manager then select UEFI Shell. The Raspberry Pi 4 boots to the UEFI Shell. Figure B-8: Boot Manager menu on page 86 shows the UEFI Shell option:

Figure B-8: Boot Manager menu

```
Raspberry Pi 4 Model B
BCM2711 (ARM Cortex-A72)                                1.50 GHz
UEFI Firmware v1.28                                     3072 MB RAM


> Console Preference Selection                              options.

  Select Language            <English>                      This selection will
                                                            take you to the
> Device Manager                                            Device Manager
> Boot Manager
> Boot Maintenance Manager

  Continue
  Reset



 ^v=Move Highlight       <Enter>=Select Entry
```

3. Use the `map` command to see if a storage device is mounted. Figure B-9: Map command output on page 86 shows a USB drive is mounted as FS0:

Figure B-9: Map command output

```
Shell> map
Mapping table
      FS1: Alias(s):HD1b:;BLK3:
          VenHw(100C2CFA-B586-4198-9B4C-1683D195B1DA)/HD(1,MBR,0x0E26C215,0x800,
0x3B71800)
      FS0: Alias(s):HD0c0b:;BLK1:
          PcieRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/USB(0x2,0x0)/HD(1,GPT,A39411BF
-9A10-41FA-9FAD-3A2303FAF354,0x800,0x13FFFF)
     BLK2: Alias(s):
          VenHw(100C2CFA-B586-4198-9B4C-1683D195B1DA)
     BLK0: Alias(s):
          PcieRoot(0x0)/Pci(0x0,0x0)/Pci(0x0,0x0)/USB(0x2,0x0)
Shell> fs0:
FS0:\>
```

4. Change the directory to FS0 by typing `FS0:` at the command prompt.

The following table shows UEFI Shell commands which are helpful for debugging:

Table B-3: Uefi Shell Commands

Command	Description
`pci`	Show PCIe devices or PCIe function configuration space information
`drivers`	Show a list of drivers
`devices`	Show a list of devices managed by EFI drivers

Command	Description
`devtree`	Show a tree of devices
`dh -d -v > dh_d_v.txt`	Save a dump of all UEFI Driver Model-related handles to dh_d_v.txt
`memmap`	Save the memory map to memmap.txt
`smbiosview`	Show SMBIOS information
`acpiview -l`	Show a list of ACPI tables
`acpiview -r 2`	Validate that all ACPI tables required by SBBR 1.2 are installed.
`acpiview -s DSDT -d`	Generate a binary file of DSDT ACPI table.
`dmpstore -all > dmpstore.txt`	Dump all UEFI variables to dmpstore.txt

See the UEFI Shell Specification for more details. The Shell commands section provides a list of shell commands, descriptions, and examples.

B.7 Set UEFI variables

You can view and change the Raspberry Pi 4 UEFI configuration settings using the UI configuration menu and UEFI shell. To configure the Raspberry Pi 4 using the UEFI Shell, use `setvar` to read and write the UEFI variables for the GUID CD7CC258-31DB-22E6-9F22-63B0B8EED6B5.

To read a setting, use the following command:

```
setvar <NAME> -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5
```

To write a setting, use the following command:

```
setvar <NAME> -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5 -bs -rt -nv =<VALUE>
```

For string-type settings such as Asset Tag, use the following command:

```
setvar <NAME> -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5 -bs -rt -nv =L"<VALUE>"
 =0x0000
```

The following commands are examples of reading and modifying UEFI variables:

Read the System Table Selection setting

```
Shell> setvar SystemTableMode -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5
```

Change the System Table Selection setting to Devicetree

```
Shell> setvar SystemTableMode -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5 -bs -rt -nv
 =0x00000002
```

Read the Limit RAM to 3 GB setting:

```
Shell> setvar RamLimitTo3GB -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5
```

Change the Limit RAM to 3 GB setting to Disabled:

```
Shell> setvar RamLimitTo3GB -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5 -bs -rt -nv
 =0x00000000
```

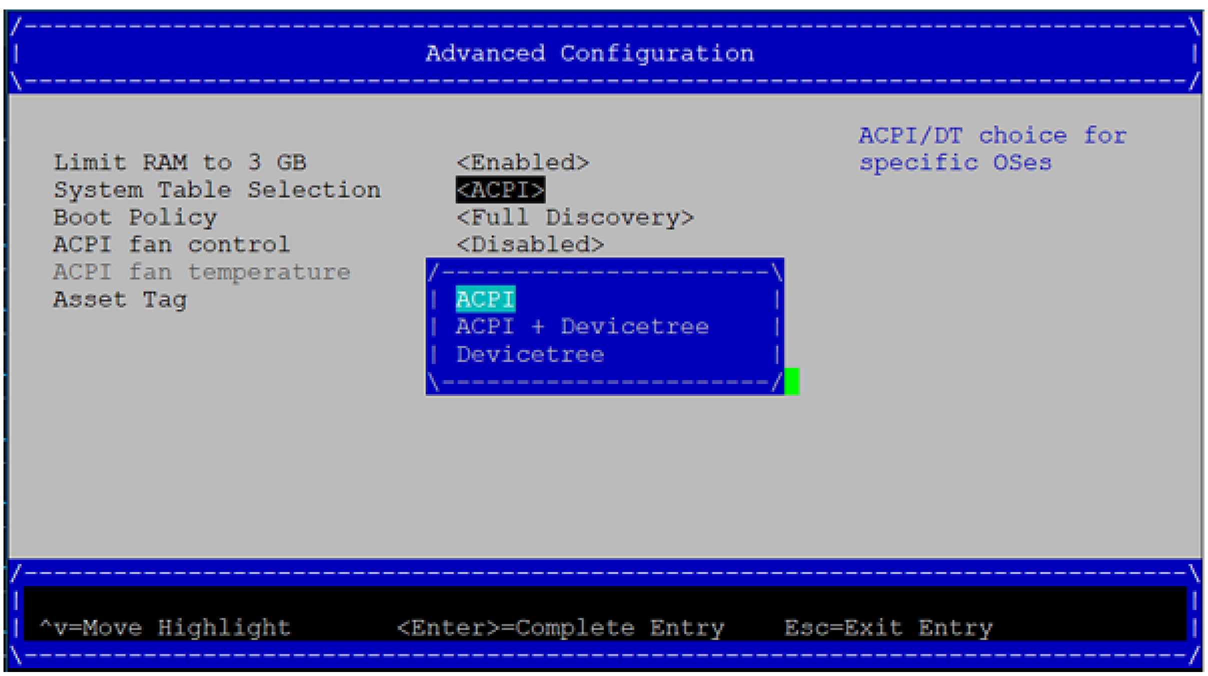
Change the Asset Tag to the string ASSET-TAG-123:

```
Shell> setvar AssetTag -guid CD7CC258-31DB-22E6-9F22-63B0B8EED6B5 -bs -rt -nv
 =L"ASSET-TAG-123" =0x0000
```

B.8 Set the system table selection

In the **Advanced Configuration** menu, select ACPI as shown in Figure B-10: ACPI option on page 88:

Figure B-10: ACPI option



B.9 Set the console preference

Linux uses the `/chosen/stdout-path` DT property or the SPCR ACPI table to show that the primary console is the serial port, even if a graphical console is available. Therefore, for some Linux OSes, set the preference to **Graphical** to remove the SPCR table which make the graphical console work.

To select the graphical console:

1. Open **Device Manager** in the UEFI menu
2. Select **Console Preference Selection**. Figure B-11: Console Preference Selection option on page 89 shows the **Console Preference Selection** option:

Figure B-11: Console Preference Selection option

1. In the **Console Preference Selection** menu, select **Graphical** or **Serial**.
2. To get serial console messages, set the preference to **Serial**.

Note

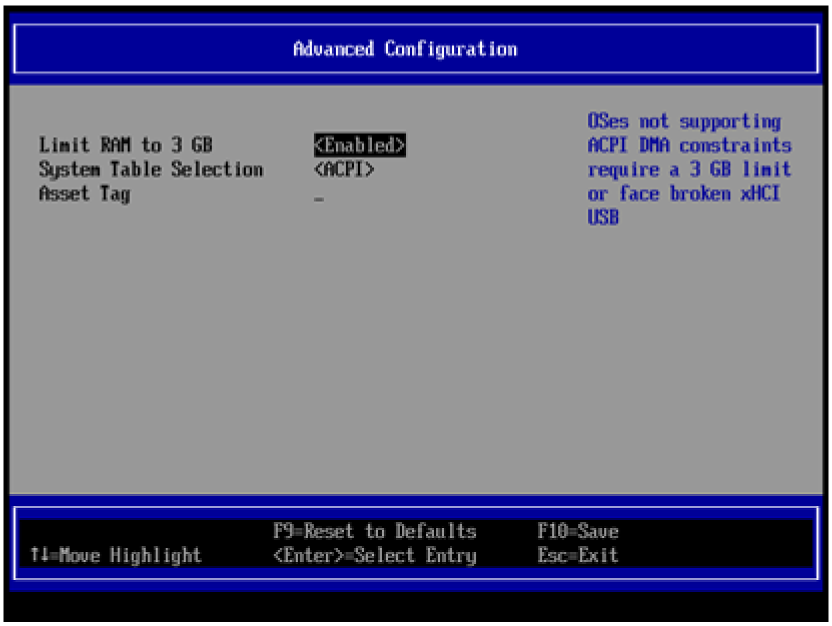
The serial console on most OSes may not work with the Graphical setting because the UEFI does not install the SPCR ACPI table. This setting must be **Serial** when running the ACS test suite because the SPCR ACPI table is mandatory for the SystemReady band and is used in parts of the ACS.

B.10 Limit RAM to 3GB

Currently, many operating systems support 3GB of RAM on the Raspberry Pi 4. To set the limit to 3GB:

1. From the UEFI menu go to **Device Manager** > **Raspberry Pi Configuration** > **Advanced Configuration**
2. Enable Limit RAM to 3GB. Figure B-12: RAM limit enabled on page 90 shows the RAM limit setting:

Figure B-12: RAM limit enabled



The following operating systems do not require a 3GB RAM limit:

- OpenBSD 6
- NetBSD 9
- VMWare ESXi