Arm® SBSA Architecture Compliance

Revision: r3p0

User Guide

101547_0300_01_en

Arm® SBSA Architecture Compliance

User Guide

Release Information

Document History

Issue	Date	Confidentiality	Change
A	30 November 2016	Non-Confidential	Alpha release
B	31 March 2017	Non-Confidential	Beta release
C	13 July 2017	Non-Confidential	REL 1.0
D	11 May 2018	Non-Confidential	REL 2.0
0200-01	27 December 2018	Non-Confidential	REL 2.1. The document now follows a new numbering format.
0200-02	26 April 2019	Non-Confidential	REL 2.2
0200-03	18 September 2019	Non-Confidential	REL 2.3
0200-04	20 March 2020	Non-Confidential	REL 2.4
0300-01	30 September 2020	Non-Confidential	REL 3.0

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

developer.arm.com

Contents
Arm® SBSA Architecture Compliance User Guide

Preface

This preface introduces the *Arm® SBSA Architecture Compliance User Guide*.

It contains the following:
- *About this book* on page 6.

About this book

This book is the user guide for Arm® SBSA architecture compliance.

Using this book

This book is organized into the following chapters:

Chapter 1 UEFI shell application
This chapter provides information on executing tests from the UEFI Shell application.

Chapter 2 Linux application
This chapter provides information on executing tests from the Linux application.

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

Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the *Arm® Glossary* for more information.

Typographic conventions

italic
Introduces special terminology, denotes cross-references, and citations.

bold
Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`
Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`monospace`
Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`
Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`
Denotes language keywords when used outside example code.

`<and>`
Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS
Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Additional reading

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

Arm publications

- *Arm® Server Base System Architecture Specification* (ARM-DEN-0029 Version 6.0).
- *Arm® Server Base Boot Requirements* (ARM-DEN-0044B).
- *Arm® Architecture Reference Manual ARMv8, for Armv8-A architecture profile* (ARM DDI 0487F.a (ID021920)).
- *Arm® Generic Interrupt Controller Architecture Specification for GIC architecture version 3.0 and version 4.0* ARM IHI 0069C (ID070116).
- *GICv3 and GICv4 Software Overview* (DAI 0492).

Other publications

Feedback

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to *support-enterprise-acs@arm.com*. Give:

- The title *Arm SBSA Architecture Compliance User Guide*.
- The number 101547_0300_01_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information

- *Arm® Developer*.
- *Arm® Documentation*.
- *Technical Support*.
- *Arm® Glossary*.

Chapter 1
UEFI shell application

This chapter provides information on executing tests from the UEFI Shell application.

It contains the following sections:

1.1 Overview of tests

The general division of tests between UEFI Shell application and Linux application is illustrated in the following table.

Table 1-1 Test environment and modules

Test environment	Modules
UEFI Shell	PE, GIC, Timers, Watchdog, Wakeup, Secure devices, PCIe, NIST
Linux command line	PCIe, SMMU, Exerciser
Bare-metal	Exerciser

1.2 UEFI application arguments

Run the UEFI Shell application with the following set of arguments:

```
uefi shell> sbsa.efi [-v <n>] [-l <n>] [-skip <x,y,z>] [-f <file name>] [-s] [-p <n>] [-nist]
```

Note

- After all the SBSA tests are run and the test results are printed on the UEFI console, the UEFI session becomes unusable.
- The UEFI Shell application is enhanced to accept an additional argument `[-p <n>]` for PCIe. This is to enable optionally running SBSA v6.0 PCIe tests even when the other tests run at older levels. For example, you can optionally run SBSA v6.0 PCIe tests even when running other SBSA tests at level 3.

The argument descriptions are available in the following table.

Table 1-2 Descriptions of UEFI application arguments

Argument	Description
`-v`	Print level **1** INFO and above. **2** DEBUG and above. **3** TEST and above. **4** WARN and ERROR. **5** ERROR.
`-l`	Level of compliance to be tested for (0-6). The default value is 4.
`-skip`	Overrides the suite to skip the execution of a particular test. It allows a maximum of three values (comma-separated). For example, 300 skips test case with ID = 300. 500 skips all tests in module with ID = 500. For details on module IDs, see *1.3 Test IDs* on page 1-12.
`-f`	File name to which the output log is written.
`-s`	Runs Secure tests before executing Non-secure tests. It requires Secure firmware code from SBSA ACS to be ported to EL3 FW. If this option is not provided, only Non-secure tests are run.
`-p`	Enables or disables the execution of SBSA v6.0 PCIe compliance tests (RCiEP rules). Allowed values for `<n>` are 0 and 1. 1 enables PCIe tests and 0 disables PCIe tests. **Note** • If this option is not provided, SBSA v6.0 PCIe (RCiEP rules) tests are not run. • If `-l` has a value of 4 and above, these tests are always run.
`-nist`	Runs SBSA ACS with NIST STS.

Example

```
shell > sbsa.efi –v 2 –l 3 –skip 20,36 –f acs.txt -p 1
```

The set of parameters shown in the code block:

- Prints messages with verbosity of 2 and above.
- Tests for compliance against SBSA level 3 for other tests and runs SBSA v6.0 PCIe (RCiEP rules) tests.
- Skips execution of all tests belonging to GIC module and test number 36.
- Stores the log messages to the file `acs.txt`.

1.3 Test IDs

The test ID of each test is generated as an addition of module ID and unit test ID.

For a given module, unit test ID begins from 1. Module IDs are as follows.

Table 1-3 Module Name and Module ID

Module name	Module ID
PE	0
GIC	100
Timer	200
Watchdog	300
PCIe	400
Power and Wakeup	500
Peripheral	600
SMMU	700
Exerciser	800
Secure	900
NIST	1000

1.4 UEFI implementation of PAL APIs

The following table lists the UEFI interfaces used for the implementation of the Platform Abstraction Layer (PAL) APIs mentioned in the *Arm® SBSA Architecture Compliance Validation Methodology* document. PAL APIs are classified into infrastructure and module-specific APIs.

Infrastructure APIs

Table 1-4 PAL APIs and UEFI interfaces

PAL API	UEFI interfaces
`pal_print`	AsciiPrint
`mem_alloc`	gBS->AllocatePool
`mem_free`	gBS->FreePool
`mem_alloc_shared`	gBS->AllocatePool
`mem_free_shared`	gBS->FreePool
`mem_get_shared_addr`	None
`mmio_read`	None
`mmio_write`	None

Module-specific APIs

Table 1-5 PAL APIs, UEFI interfaces, and ACPI tables consumed

PAL API	UEFI interfaces consumed	ACPI table consumed
`pe_create_info_table`	• gST->ConfigurationTable • CompareGuid • IndustryStandard/Acpi61.h	MADT Table
`call_smc`	None	-
`pe_execute_payload`	None	-
`pe_install_esr`	• gEfiCpuArchProtocolGuid • Cpu->RegisterInterruptHandler	-
`gic_create_info_table`	• gST->ConfigurationTable • CompareGuid • IndustryStandard/Acpi61.h	MADT table
`gic_install_isr`	• gHardwareInterruptProtocolGuid • RegisterInterruptSource • EnableInterruptSource	-
`timer_create_info_table`	• gST->ConfigurationTable • CompareGuid • IndustryStandard/Acpi61.h	GTDT table
`wd_create_info_table`	• gST->ConfigurationTable • CompareGuid • IndustryStandard/Acpi61.h	GTDT table
`pcie_create_info_table`	• gST->ConfigurationTable • CompareGuid • IndustryStandard/Acpi61.h	MCFG table
`pcie_get_mcfg_ecam`	• gST->ConfigurationTable • CompareGuid, IndustryStandard/Acpi61.h • IndustryStandard/ MemoryMappedConfigurationSpaceAccessTable.h	MCFG table
`iovirt_create_info_table`	• gST->ConfigurationTable • CompareGuid • IndustryStandard/Acpi61.h	IORT table
`peripheral_create_info_table`	• gEfiPciIoProtocolGuid • Pci->GetLocation • Pci->Pci.Read	-
`memory_create_info_table`	gBS->GetMemoryMap	-

Chapter 2
Linux application

This chapter provides information on executing tests from the Linux application.

It contains the following sections:

2.1 Linux application arguments

Run the Linux application with the following set of arguments:

```
shell> sbsa [--v <n>] [--l <n>] [--skip <x,y,z>]
```

Table 2-1 Description of Linux application arguments

Argument	Description
v	Print level **1** INFO and above **2** DEBUG and above **3** TEST and above **4** WARN and ERROR **5** ERROR
l	Level of compliance to be tested for. (0 to 6)
skip	Overrides the suite to skip the execution of a particular test. For example, 53 skips test case with ID 53.

Example

```
shell> sbsa --v 3 –-l 3 --skip 57
```

This set of parameters tests for compliance against SBSA level 3 with print verbosity set to 3, and skips test number 57.

Loading the kernel module

Before the SBSA ACS Linux application can be run, load the SBSA ACS kernel module using the `insmod` command.

```
shell> insmod sbsa_acs.ko
```

2.2 Build steps and environment setup

This section lists the porting and build steps for the kernel module.

The patch for the kernel tree and the Linux PAL are hosted separately on *linux-arm.org*.

Building the kernel module

Prerequisites

- Linux kernel source version 4.14.
- Linaro GCC tool chain 5.3 or above.
- Build environment for AArch64 Linux kernel.

Porting steps for Linux kernel

1. `git clone git://linux-arm.org/linux-acs.git <local_dir/sbsa-acs-drv>`
2. `git clone https://github.com/ARM-software/sbsa-acs.git <local_dir/sbsa-acs>`
3. Apply the `<local_dir>/kernel/src/0001-Enterprise-acs-linux-v4.13.patch` patch to your kernel source tree.
4. Build the kernel.

Build steps for SBSA kernel module

1. `cd <local_dir>/sbsa-acs-drv/files`
2. Set `CROSS_COMPILE` to the ARM64 toolchain path.
3. `export KERNEL_SRC=<linux kernel path>`
4. `./setup.sh <local_dir/sbsa-acs>`
5. `./linux_sbsa_acs.sh`

`sbsa_acs.ko` file is generated.

SBSA Linux application build

1. `cd <sbsa-acs path>/linux_app/sbsa-acs-app`
2. Set `CROSS_COMPILE` to the ARM64 toolchain path.

```
export CROSS_COMPILE=<local_dir>/gcc-linaro-5.3-2016.02/bin/aarch64-linux-gnu-
```

3. `make`

The executable file `sbsa` is generated.

This section contains the following subsections:

- *2.2.1 Target environment setup* on page 2-17.
- *2.2.2 Runtime environment* on page 2-18.

2.2.1 Target environment setup

The set of tests assumes that at least one SATA controller is behind a PCIe root complex. The SATA controller may or may not be behind an IOMMU.

Before running these tests, at least one SATA hard disk must be connected to the SATA controller. The test performs read and write operations to the SATA hard disk. Therefore, the data on the HDD is overwritten. The SATA drive must not be the boot device for the OS.

2.2.2 Runtime environment

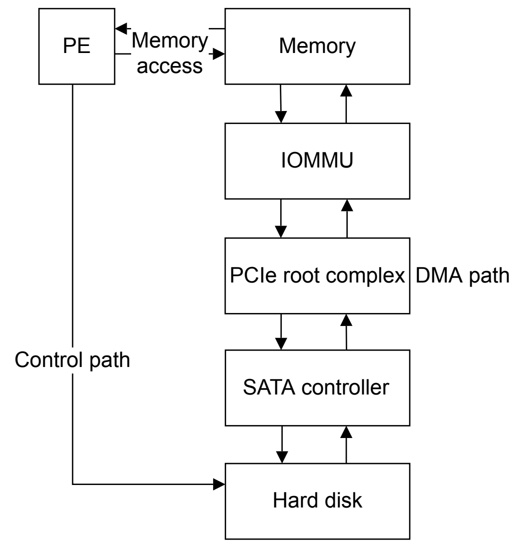

Figure 2-1 Hardware functional blocks

The PCIe-DMA tests initiate data transfers from a DMA master. By default, the test searches for a SATA controller which is part of the PCIe subsystem.

1. The test writes known data from the PE to main memory.
2. The test programs the DMA master to transfer this known data to its end-point device.
3. The test asks the DMA master to transfer the data back to a different location in the main memory.
4. The test compares the data at both the locations.

If the SATA controller is not behind an IOMMU, during this data transfer, the address that is used by the SATA controller is retrieved and compared with the DMA address that is seen by the PE.

If the DMA master is behind an IOMMU, then the address that is used by the SATA AHCI controller is compared with the address that is seen by the IOMMU. Both these addresses must match.

To enable the export of the addresses that are seen by the SATA AHCI controller and IOMMU, the kernel drivers for these two modules must be patched.

Appendix A
Revisions

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

It contains the following section:

A.1 Revisions

Table A-1 Issue0200-01

Change	Location
Information about exerciser is added.	See *1.3 Test IDs* on page 1-12.
A new parameter `[--e]` is added to Linux application arguments.	See *2.1 Linux application arguments* on page 2-16.

Table A-2 Differences between Issue 0200-01 and Issue 0200-02

Change	Location
Bare-metal test environment is added to the table.	See *1.1 Overview of tests* on page 1-9.
A note about additional porting for the exerciser is added.	See *2.1 Linux application arguments* on page 2-16.

Table A-3 Differences between Issue 0200-02 and Issue 0200-03

Change	Location
No technical changes.	-

Table A-4 Differences between Issue 0200-03 and Issue 0200-04

Change	Location
• Arguments for NIST and PCIe tests are added. • A note about UEFI session is added.	See *1.2 UEFI application arguments* on page 1-10.
NIST module ID is updated.	See *1.3 Test IDs* on page 1-12.
Linux application arguments are updated.	See *2.1 Linux application arguments* on page 2-16.

Table A-5 Differences between Issue 0200-04 and Issue 0300-01

Change	Location
Additional level of compliance to be tested is added.	See table in *1.2 UEFI application arguments* on page 1-10 and *2.1 Linux application arguments* on page 2-16.