Arm® SBSA Architecture Compliance Suite

User Guide

Non-Confidential – DEV2.0

PJDOC-2042731200-3438

Arm® SBSA Architecture Compliance Suite
User Guide

Release Information

Document History

Date	Confidentiality	Change
30 November 2016	Non-Confidential	Release for alpha
31 March 2017	Non-Confidential	Release for beta
13 July 2017	Non-Confidential	Release for v1.0
11 May 2018	Non-Confidential	Release for DEV2.0

Proprietary Notice

Confidentiality Status

Product Status

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

Web Address

http://www.arm.com

Contents

Arm® SBSA ACS User Guide

Preface

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

It contains the following sections:

About this book

This book describes the Architecture Compliance Suite User Guide for the Arm® SBSA architecture.

Product revision status

The r*m*p*n* identifier indicates the revision status of the product described in this book, for example, r*1*p*2*, where:

r*m* Identifies the major revision of the product, for example, r1.

p*n* Identifies the minor revision or modification status of the product, for example, p2.

Intended audience

This book is written for engineers who are designing or verifying an implementation of the Arm® SBSA architecture.

Using this book

This book is organized into the following chapters:

Chapter 1 UEFI shell application

Read this for information about SBSA Architecture Compliance Suite tests which run as a UEFI Shell Application.

Chapter 2 Linux kernel module and application

Read this for details about SBSA Architecture Compliance Suite tests which run within an Operating System environment.

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` (with `mono` underlined)

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.

Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

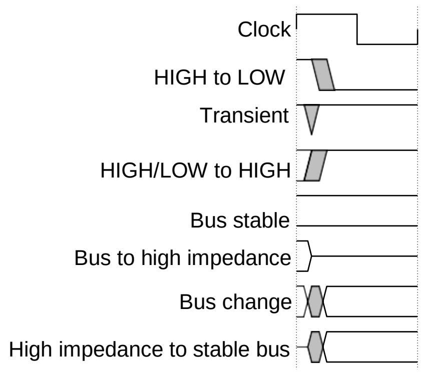

Figure 1 Key to timing diagram conventions

Signals

The signal conventions are:

Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lowercase n

At the start or end of a signal name denotes an active-LOW signal.

Additional reading

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

Arm Publications

- *Server Base System Architecture (ARM-DEN-0029 Version 3.0)*
- *Server Base Boot Requirements (ARM-DEN-0044B)*
- *Arm® Architecture Reference Manual ARMv8, for Armv8-A architecture profile (ARM DDI 0487).*

Other publications

None.

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 Suite User Guide*.
- The number PJDOC-2042731200-3438.
- 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.

Chapter 1
UEFI Shell application

This chapter contains the following sections:

1.1 Overview of tests

The general division of tests between UEFI Shell application and Linux application is as follows.

Table 1-1 Test environment and components covered

Test environment	Components covered
UEFI Shell	PE, GIC, Timers, Watchdog, Wakeup, Secure Devices
Linux command line	PCIe, SMMU

1.2 UEFI application arguments

The application can be run with the following set of arguments:

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

Table 1-2 Parameters and descriptions

Parameter	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 (0-5).
`skip`	Overrides the suite to skip the execution of a particular test. Example 33 skips test case with ID 33. 30 skips all tests in module with ID = 30. 50 skips all tests in module with ID = 50. (Refer to test ID section for more details on Module IDs) comma separated. Maximum of three values.
`f`	File name to which the output log is written.
`s`	Runs Secure tests before executing Non-secure tests. (Requires Secure firmware code from SBSA ACS to be ported to EL3 FW) Not giving this option runs only Non-secure tests.

For example,

```
Shell > sbsa.efi –v 2 –l 3 –f acs.txt –skip 20,36
```

This set of parameters:

- Prints messages with verbosity of 2 and above.
- Tests for compliance against SBSA level 3.
- Skips execution of all tests belonging to GIC module and test number 36.
- Stores the log messages to file `acs.txt`.

1.3 Memory requirements

Code

Binary size – 165KB

Data

EfiBootServicesData

Table 1-3 Data Structure and Size

Data Structure	Size (in Bytes)
PE_INFO_TABLE	8192
GIC_INFO_TABLE	2048
TIMER_INFO_TABLE	1024
WD_INFO_TABLE	512
PCIE_INFO_TABLE	64
PERIPHERAL_INFO_TABLE	1024
IO-Virtualization Table	2048
PE_SHARED_MEMORY	(num_of_pe) * 16
PE_SECONDARY_STACK	(num_of_pe) * 256
Total (Assuming 48 PEs)	**27,992**

EfiRuntimeServicesData

None

1.4 Interfaces consumed by shell application

Libraries

- `UefiLib`
- `ShellLib`
- `BaseMemoryLib`
- `ShellCEntryLib`
- `UefiBootServicesTableLib`
- `UefiRuntimeServicesTableLib`

Protocols

- `gEfiAcpiTableProtocolGuid`
- `gHardwareInterruptProtocolGuid`
- `gEfiPciIoProtocolGuid`

1.5 Toolchain

Linaro AArch64 5.3 toolchain was used to develop this application.

The toolchain is at:

http://releases.linaro.org/components/toolchain/binaries/5.3-2016.02/aarch64-linux-gnu/

1.6 System dependencies

PSCI

The compliance suite makes the following PSCI calls:

ARM_SMC_ID_PSCI_CPU_SUSPEND_AARCH64	(0xc4000001)
ARM_SMC_ID_PSCI_CPU_OFF	(0x84000002)
ARM_SMC_ID_PSCI_CPU_ON_AARCH64	(0xc4000003)

1.7 Platform override

On certain platforms, the underlying ACPI infrastructure to provide information on the system is not implemented yet. To enable running SBSA ACS on these platforms, override hooks are provided for certain modules which take the relevant hardware information from the override file rather than the underlying UEFI framework.

See `<acs_local_path>/sbsa-acs/platform/pal_uefi/include/platform_override.h` file in the source code for available options.

1.8 Test ID

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 1-4 Module Name and Module ID

Module Name	Module ID
PE	0
GIC	20
Timer	30
Watchdog	40
PCIe	50
Power & Wakeup	70
Peripheral	80
SMMU	90
Secure	900

1.9 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 *SBSA Validation Methodology Document.* See https://github.com/ARM-software/sbsa-acs/tree/master/docs/SBSA_Val_Methodolgy.pdf

1.9.1 Infrastructure APIs

Table 1-5 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 1-6 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	MCFG Table

PAL API	UEFI interfaces consumed	ACPI table consumed
	• IndustryStandard/MemoryMappedConfigurationSpaceAccessTable.h	
`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 kernel module and application

This chapter contains the following sections:

2.1 Linux application arguments

The application can be run with the following set of arguments:

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

Table 2-1 Parameter and its description

Parameter	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 5)
skip	Overrides the suite to skip the execution of a particular test. Example 53 skips test case with ID 53.

For 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.

2.1.1 Loading the kernel module

Before the SBSA ACS Linux application can be run, the SBSA ACS kernel module must be loaded. This can be achieved by using the `insmod` command.

For example,

```
Shell>insmod sbsa_acs.ko
```

2.2 SBSA ACS - Linux application

2.2.1 Source path

The patch for the kernel tree and the Linux Platform Abstraction Layer are hosted separately on http://linux-arm.org/git?p=linux-acs.git;a=summary.

2.2.2 Kernel module – build

Prerequisites

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

Porting steps – 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`](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 – 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.

2.2.3 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

Executable file: `sbsa` is generated.

2.2.4 Target environment setup

The set of tests assumes at least one SATA controller 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.

2.2.5 Runtime environment

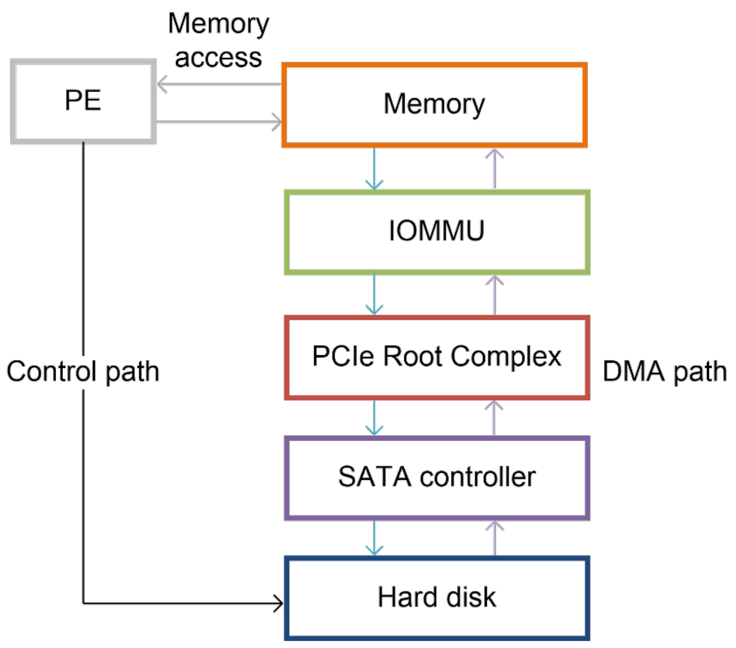

Figure 2-1 Hardware functional blocks

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.

Also, 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.