ARM® SBSA Architecture Compliance

r0p1

Validation Methodology

Non-Confidential - Beta

ARM SBSA Architecture Compliance
Validation Methodology

Release Information
The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
30 Nov 2016	-	Non-Confidential	Release for alpha
31 Mar 2017	-	Non-Confidential	Release for Beta

Proprietary Notice

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Confidentiality Status

This document is non-confidential.

Product Status

The information in this document is for an beta product, that is a product under development.

Web Address

http://www.arm.com

Contents
ARM SBSA Architecture Compliance – Validation Methodology

Preface

This preface introduces the *ARM® SBSA Architecture Compliance - Validation Methodology*. It contains the following sections:

About this book

This book describes the Architecture Compliance - Validation Methodology for the ARM SBSA architecture.

Intended audience

This book is written for engineers who are specifying, designing, or verifying an implementation of the ARM SBSA architecture.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
Read this for an overview of SBSA validation methodology.

Chapter 2 Execution model and flow control
Read this for details on the execution control mode and flow control scheme used by the compliance and for details on the different test platform variants and ideas on implementing the PAL layer.

Conventions

The following sections describe conventions that this book can use:

- *Typographic conventions*.
- *Signals*.
- *Numbers*.
-

Typographic conventions

The typographical conventions are:

italic	Introduces special terminology, and denotes citations.
bold	Denotes signal names, and is used for terms in descriptive lists, where appropriate.
monospace	Used for assembler syntax descriptions, pseudocode, and source code examples. Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.
SMALL CAPITALS	Used for a few terms that have specific technical meanings.
Colored text	Indicates a link. This can be: • A URL, for example http://infocenter.arm.com. • A cross-reference, that includes the page number of the referenced information if it is not on the current page, for example, *Feedback on this product* on page xi. • A link, to a chapter or appendix, or to the section of the document that defines the colored term, for example Appendix A *Memory Attributes*.

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by 0b, and hexadecimal numbers by 0x. In both cases, the prefix and the associated value are written in a monospace font, for example 0xFFFF0000.

Additional reading

This document refers to the following external documents.

- http://www.uefi.org/specifications
  - ACPI Specification Version 6.1
  - UEFI Specification Version 2.6
  - UEFI Platform Initialization Specification Version 1.4 (Volume 2)
- http://www.uefi.org/acpi
  - WDRT
  - MCFG
  - IORT
  - DBG2

See Infocenter: http://infocenter.arm.com for access to ARM documentation

ARM Publications

This document refers to the following documents:

- *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).*

Chapter 1
Introduction

This chapter contains the following sections:

1.1 About the SBSA

The Server Base Systems Architecture (SBSA) specification specifies hardware system architecture, based on the ARM 64-bit architecture, which the server system software, such as operating systems, hypervisors and firmware can rely on. It addresses PE features and key aspects of system architecture.

The primary goal is to ensure enough standard system architecture to enable a suitably-built single OS image to run on all hardware compliant with this specification.
It also specifies features that firmware can rely on, allowing for some commonality in firmware implementation across platforms

The SBSA architecture that is described in the SBSA Architecture Specification defines the behavior of an abstract machine, referred to as an SBSA system. Implementations compliant with the SBSA architecture must conform to the described behavior of the SBSA Architecture Specification.

The Architecture Compliance Suite is a set of examples of the invariant behaviors that are specified by the SBSA Specification. Use this suite to verify that these behaviors are implemented correctly in your system.

1.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Table 1-1 Terms and meanings

Term	Meaning
ACPI	Advanced Configuration and Power Interface
GIC	Generic Interrupt Controller
SPI	Shared Peripheral Interrupt
PPI	Private Peripheral Interrupt
LPI	Locality Specific Peripheral Interrupt
XSDT	eXtended System Description Table
SMC	Secure Monitor Call
PE	Processing Element
PSCI	Power State Coordination Interface
PCIe	Peripheral Component Interconnect - Express
SBSA	Server Base Systems Architecture
UART	Universal Asynchronous Receiver/Transmitter
UEFI	Unified Extensible Firmware Interface
ELx	Exception Level ‘x’ where x can be 0,1,2 or 3

1.3 Compliance test

SBSA compliance tests are self-checking, portable C-based tests with directed stimulus. Table 1-2 describes the components of the compliance test with the description.

Table 1-2 Components of the compliance test

Components	Description
PE	Tests to verify PE compliance
GIC	Tests to verify GIC compliance
Timers	Tests to verify PE timers and System timers compliance
Watchdog	Tests to verify Watchdog compliance
PCIe	Tests to verify PCIe subsystem compliance
Peripherals	Tests to verify USB, SATA and UART compliance
Power states	Tests to verify System power states compliance
SMMU	Tests to verify SMMU subsystem compliance
Secure	Tests to verify Secure hardware

1.4 Layered Software Stack

Compliance tests use the layered software-stack approach to enable porting across different test platforms. As shown in Figure 1-1, the constituents of the layered stack are:

1. Test suite
2. Validation Abstraction Layer (VAL)
3. Platform Abstraction Layer (PAL)

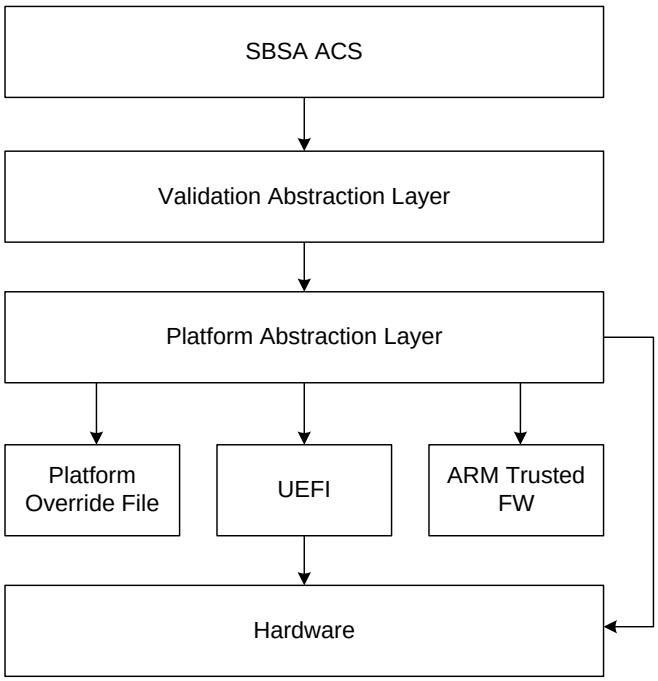

Figure 1-1 Layered software stack for compliance suite

Figure 1-2 illustrates the compliance test software-stack interplay with UEFI shell application as an example

Table 1-3 describes the different layers of compliance test.

Table 1-3 Different layers of compliance test

Layer	Description
Test Suite	This is a collection of targeted tests which validate the compliance of the target system. These tests use interfaces provided by the VAL layer.
Validation Abstraction Layer (VAL)	This layer provides a uniform view of all the underlying hardware and test infrastructure to the test suite.
Platform Abstraction Layer (PAL)	This layer abstracts features whose implementation varies from one target system to another. The PAL is a C-based API defined by ARM and implemented by you. Each test platform requires a PAL implementation of its own. The PAL APIs are meant for the compliance test to reach or use other abstractions in the test platform such as the UEFI infrastructure and bare-metal abstraction.

Figure 1-2 shows the compliance test software-stack with UEFI shell application as an example.

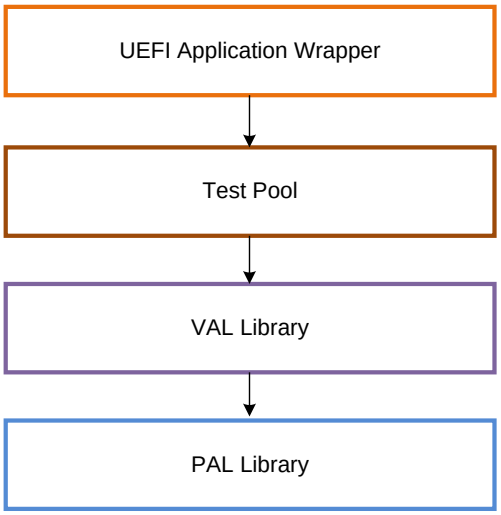

Figure 1-2 Compliance test software-stack with UEFI shell application

Figure 1-3 shows the compliance test software stack with Linux Application as an example:

The stack is spread across user-mode and kernel-mode space.

The Linux command line application running in user-mode space and the Kernel module communicate using a Sysfs interface.

The Test pool, VAL and PAL layers are built as a kernel module.

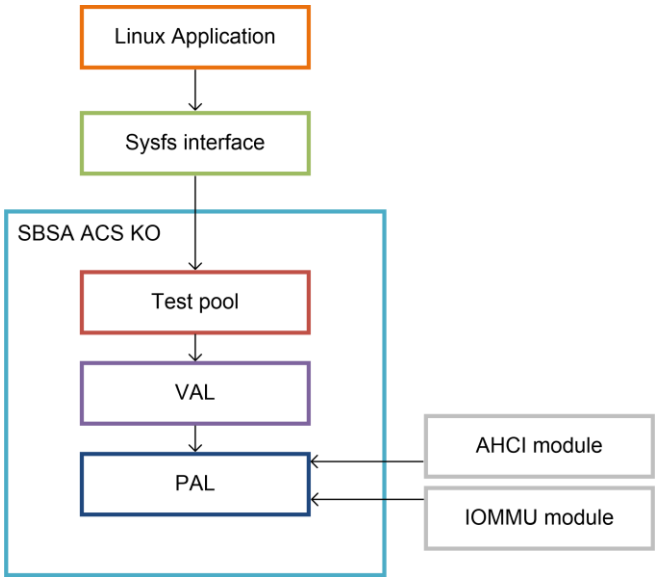

Figure 1-3 Compliance test software-stack with Linux application

The SBSA command line application initiates the tests and queries for status of the test using the standard Sysfs interface of the Linux OS.

To avoid multiple data transfers between kernel and user mode, the test suite, VAL and PAL are together built as a kernel module.

Further, the PAL layer may need information from modules such as AHCI driver and the IOMMU driver which are outside the SBSA ACS kernel module. A separate patch file will be provided to patch the drivers appropriately to export the required information. See the User Guide more details on this.

1.4.1 Coding guidelines

The coding guidelines followed for the implementation of the test suite are:

- All the tests call VAL layer APIs.
- VAL layer APIs may call PAL layer APIs depending on the functionality requested.
- A test does not directly interface with PAL layer functions.
- The Test Layer does not need any code modifications when porting from one platform to another.
- All the platform porting changes will be limited to PAL layer.
- The VAL layer may require changes if there are architectural changes impacting multiple platforms.

1.5 Test platform abstraction

The compliance suite defines and uses the test platform abstraction that is illustrated in Figure 1-3

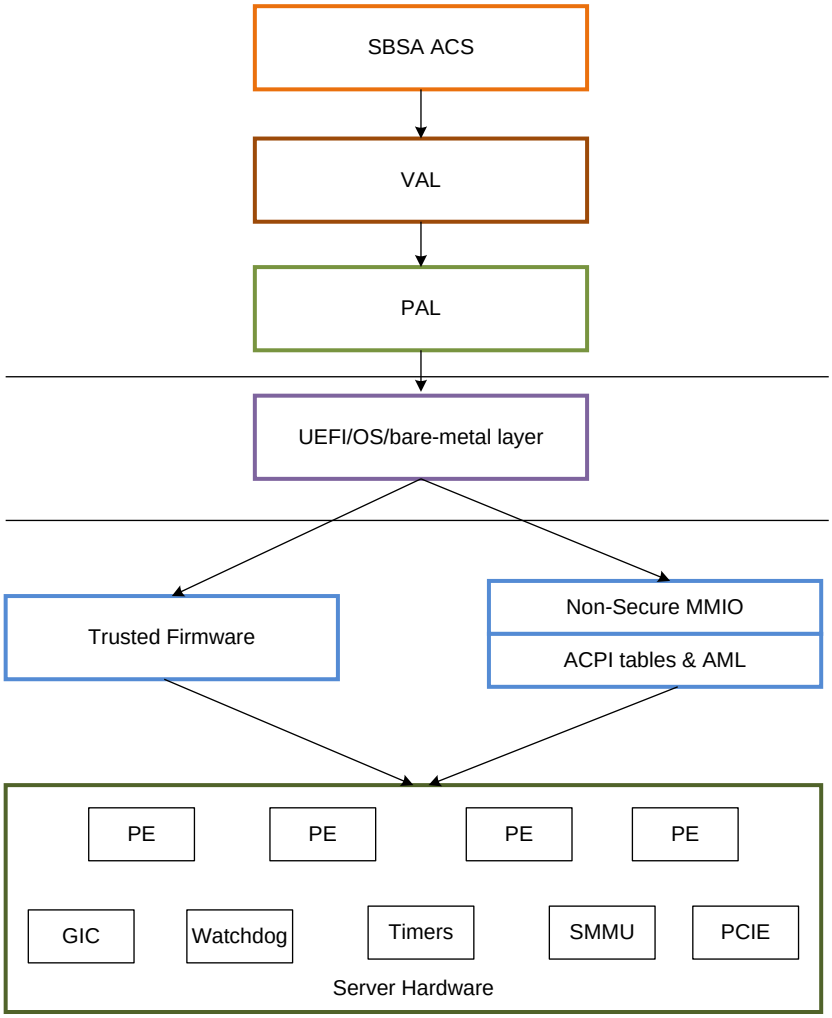

Figure 1-3 Test platform abstraction

Table 1-2 on page 1-17 describes the SBSA abstraction terms with description.

Abstraction	Description
UEFI /OS	UEFI Shell Application or Operating system provide infrastructure for console and memory management. This module runs at EL2
Trusted Firmware	Firmware which runs at EL3
ACPI	Interface layer which provides platform specific information, removing the need for the Test suite to be ported on a per platform basis
Shared Memory	Memory which is visible to all the PE and test peripherals.
Hardware	PE and controllers which are specified as part of the SBSA specification.

Chapter 2
Execution model and flow control

This chapter contains the following sections:

2.1 Execution model and flow control

The following figure describes the execution model of compliance suite and the flow control used.

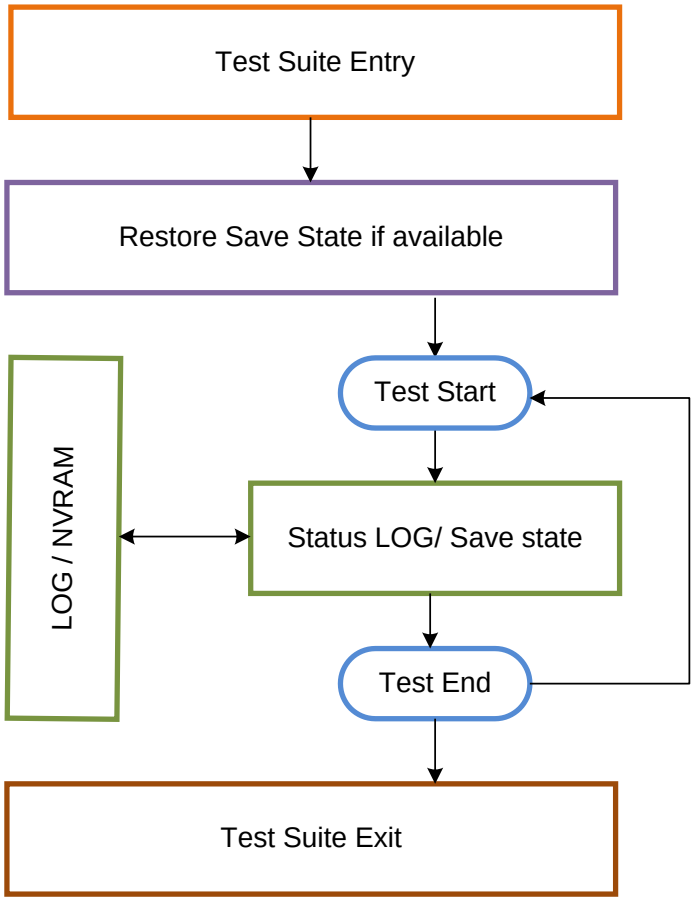

Figure 2-1 Execution model and flow control

The process followed for the flow control is as described:

Step 1. The Execution environment (for example UEFI shell) invokes the test entry point.

Step 2. The test checks if there is a saved state. In UEFI, this may be provided by UEFI variables.

Step 3. If a saved state is found, then restore the saved state (not implemented yet)

Step 4. Start the Test iteration loop.

Step 5. Save state and report status during the test execution as required.

Step 6. Reboot or put the system to sleep as required.

Step 7. Loop until all the tests are completed.

2.2 Test Build and Execution Flow

2.2.1 Pre-requisite

To build the SBSA compliance suite as a UEFI Shell application , a UEFI EDK2 source tree is required.

To Build the SBSA ACS kernel Module, Linux Kernel tree version 4.9 or above is required.

More details can be found at: https://github.com/ARM-software/sbsa-acs/blob/master/README.md

2.2.2 Source code directory

Figure 2-2 shows the source code directory for the SBSA ACS.

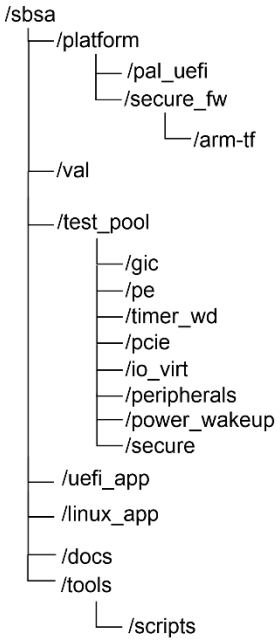

Figure 2-2 SBSA ACS directory structure

Where,

pal_uefi arm-tf	Platform code targeting UEFI Implementation ARM trusted firmware example code which needs to be integrated into the EL3 secure firmware
val uefi_app	Common code used by the tests. Makes calls to PAL as needed. UEFI application source to call into the tests entry point.
test_pool	test case source files for test suite
linux_app	Linux command line executable source code
docs	Documentation
scripts	Scripts written for this Suite.

2.2.3 Test Build - UEFI

UEFI Shell Application

The Build steps for the compliance suite to be compiled as a UEFI shell application are at https://github.com/ARM-software/sbsa-acs/blob/master/README.md

EL3 Firmware

To execute the secure tests, the EL3 firmware directory from the platform/secure_sw needs to be integrated into the Platform specific EL3 code-base.

As a reference implementation, the example code based on ARM Trusted Firmware is included as part of the ACS.

The steps to port the reference implementation and build EL3 firmware are beyond the scope of this document.

2.2.4 Test Build – OS based tests

Linux Application

The Build steps for the Linux application driven compliance suite are detailed within the User Guide at https://github.com/ARM-software/sbsa-acs/tree/master/docs.

Linux Kernel Module

The Build steps for the SBSA ACS kernel module which is a dependency for the SBSA-ACS Linux application are also part of the User Guide.