

Arm® TBSAv8-M Architecture Test

Revision: r0p0

Validation Methodology and User Guide



Arm® TBSAv8-M Architecture Test

Validation Methodology and User Guide

Copyright © 2018 Arm Limited or its affiliates. All rights reserved.

Release Information

Document History

Issue	Date	Confidentiality	Change
PJDOC-2042731200-3327	March 2018	Confidential	Pre alpha release
PJDOC-2042731200-3327	March 2018	Confidential	Alpha release
0000-01	15 June 2018	Non-Confidential	Dev0.5 release. Note: The document now follows a new numbering format.
0000-02	29 June 2018	Non-Confidential	Dev0.6 release.

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2018 Arm Limited (or its affiliates). All rights reserved.

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 for an Alpha product, that is a product under development.

Web Address

<http://www.arm.com>

Contents

Arm® TBSAv8-M Architecture Test Validation Methodology and User Guide

Preface

<i>About this book</i>	7
<i>Feedback</i>	9

Chapter 1

Introduction

1.1	<i>Abbreviations</i>	1-11
1.2	<i>TBSA-v8M test suite</i>	1-12
1.3	<i>Components of the test suite</i>	1-13
1.4	<i>Compliance sign-off process</i>	1-14
1.5	<i>Getting started</i>	1-15
1.6	<i>Feedback, contributions, and support</i>	1-17

Chapter 2

Porting steps

2.1	<i>Prerequisites for running DPM related tests in TBSA-v8M test suite</i>	2-19
-----	---	------

Chapter 3

Architecture test suite

3.1	<i>Test layering details</i>	3-21
3.2	<i>Build flow</i>	3-22
3.3	<i>Test execution flow</i>	3-23
3.4	<i>Test dispatcher</i>	3-25
3.5	<i>Test naming conventions</i>	3-27
3.6	<i>Test status reporting</i>	3-28

A.1 *Revisions* Appx-A-30

Preface

This preface introduces the *Arm® TBSAv8-M Architecture Test Validation Methodology and User Guide*.

It contains the following:

- *About this book* on page 7.
- *Feedback* on page 9.

About this book

This book describes the TBSAv8-M Architecture Test Validation Methodology.

Product revision status

The *rm**pn* identifier indicates the revision status of the product described in this book, for example, r1p2, where:

rm Identifies the major revision of the product, for example, r1.

pn Identifies the minor revision or modification status of the product, for example, p2.

Intended audience

This user guide is written for engineers who are validating an implementation of the Trusted Base System Architecture Test Suites for Armv8-M.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction

Read this chapter for an introduction to the features and components of the Trusted Base System Architecture Test Suites for Armv8-M.

Chapter 2 Porting steps

Read this chapter for information on configuring the test suite.

Chapter 3 Architecture test suite

Read this chapter for information on the tests that are provided with the test suite.

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.

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.

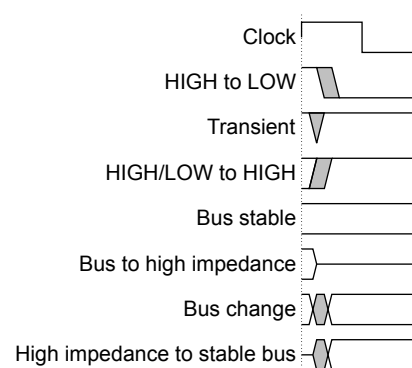


Figure 1 Key to timing diagram conventions

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

- *Arm® Platform Security Architecture Trusted Base System Architecture for Armv8-M* version Beta-1 () DEN 0062A.
- *Arm®v8-M Architecture Reference Manual* (ARM DDI 00553A.b).

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 errata@arm.com. Give:

- The title *Arm TBSAv8-M Architecture Test Validation Methodology and User Guide*.
- The number 101308_0000-0.6dev_02_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.

Chapter 1

Introduction

Read this chapter for an introduction to the features and components of the Trusted Base System Architecture Test Suites for Armv8-M.

It contains the following sections:

- [1.1 Abbreviations](#) on page 1-11.
- [1.2 TBSA-v8M test suite](#) on page 1-12.
- [1.3 Components of the test suite](#) on page 1-13.
- [1.4 Compliance sign-off process](#) on page 1-14.
- [1.5 Getting started](#) on page 1-15.
- [1.6 Feedback, contributions, and support](#) on page 1-17.

1.1 Abbreviations

This section lists the acronyms that are used in this document.

Table 1-1 Abbreviations and expansions

Abbreviation	Expansion
AES	Advanced Encryption Standard
DPM	Debug Protection Mechanism
I2C	Inter-Integrated Circuit
IDAU	Implementation Defined Attribution Unit
MPC	Memory Protection Controller
MPU	Memory Protection Unit
NSC	Non-secure Callable
NVIC	Nested Vector Interrupt Controller
NVM	Non-Volatile Memory
OTP	One-time Programmable
PAL	Platform Abstraction Layer
PE	Processing Element
PPC	Peripheral Protection Controller
SAU	Security Attribution Unit
SPI	Serial Peripheral Interface
TBSA	Trusted Base System Architecture
VAL	Validation Abstraction Layer

1.2 TBSA-v8M test suite

The test suite checks whether an implementation conforms to the behaviors described in the TBSA specifications.

The TBSA-v8M that is described in the TBSA-v8M specification defines the behavior of an abstract machine, referred to as a TBSA-v8M system. Implementations compliant with TBSA-v8M architecture must conform to the described behavior of the TBSA-v8M System.

The test suite includes the following examples:

1. Invariant behaviors that are provided by the TBSA-v8M Architecture Specification. You can use these examples to verify if these behaviors are interpreted correctly.
2. Areas of the architecture that are fundamental, known pitfalls, and common misinterpretations. The tests are not exhaustive and implementers are responsible for device verification.

The TBSA-v8M test suite contains basic and assisted architecture tests. The system designers building a basic implementation of TBSA-v8M architecture must show compliance using the basic architecture test suites. System designers building an assisted implementation of TBSA-v8M must show compliance using both the basic and assisted architecture test suites.

The test suites contain self-checking tests. These tests have checks that are embedded within the test code. Tests are coded in assembly and C.

To facilitate test reporting and management of observing aspects, the TBSA-v8M system must contain at least one UART for printing the status of tests.

This section contains the following subsection:

- [1.2.1 Scope of the document on page 1-12.](#)

1.2.1 Scope of the document

This document describes the layers of TBSA-v8M test suite and its usage. This document is intended to solicit feedback from partners so that TBSA-v8M test suite can be used agnostic to various system implementations.

Since TBSA-v8M test suite is at Alpha stage, only a subset of tests released with this suite are validated in the Arm internal platform. There are possibilities to have tests where the test code is complete for a particular scenario but cannot be validated in the Arm internal platform.

1.3 Components of the test suite

The test suite consists of the components that are described in the following table.

Components	Description
Suites	The suites are organized to align with the features of the TBSA-v8M architecture. These suites contain self-checking tests that are written in C language.
Substructure	Test supporting layers consist of a framework and libraries setup as: <ul style="list-style-type: none">• Scripts to build the test suites• VAL library• PAL library
Documentation	Kit-specific documents.

1.4 Compliance sign-off process

More details on the compliance sign-off or certification process and expectations from partner on this process will be published in the upcoming releases of this document.

1.5 Getting started

This section provides an overview about getting started with the test suite.

This section contains the following subsections:

- [1.5.1 Directory structure on page 1-15.](#)
- [1.5.2 Software, tools, and licensing requirements on page 1-16.](#)
- [1.5.3 Environment requirements on page 1-16.](#)

1.5.1 Directory structure

Validation tests require that the components of the test suite are in a specific hierarchy.

When the test suite release package is downloaded from GitHub, the top-level directory contains the files that are shown in the following figure.

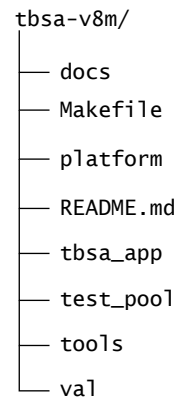


Figure 1-1 Test suite directory structure

docs

This directory contains the test suite documentation.

platform

This directory contains files to form PAL. PAL is the closest to hardware and is aware of underlying hardware details. Since this layer interacts with hardware, it is ported or tailored to specific hardware required for system components present in a platform. This layer is also responsible for presenting a consistent interface to the validation abstraction layer required for the tests.

tbsa_app

This directory contains the entry point for TBSA-v8M test suites. It is expected from partner that the System under test – boot software would give control to the TBSA-v8M test suite application entry point in Secure privileged mode.

test_pool

This directory contains the test suites. This test suite is a set of C-based directed tests, each of which verifies the implementation against a test scenario that is described by the TBSA-v8M specification. These tests are abstracted from the underlying hardware platform by the VAL.

tools

This directory contains subdirectories for the tools and scripts that are used in the test suite.

val

This directory contains subdirectories for the VAL libraries. This layer provides a uniform and consistent view of the available test infrastructure to the tests in the test pool. The VAL makes appropriate calls to the PAL to achieve this functionality. This layer is not ported when the underlying hardware changes.

1.5.2 Software, tools, and licensing requirements

The test suite requires specific versions of software to operate. The following list provides the details of the platform and tools that are needed for the test suite.

- Host Operating System: Ubuntu, Cygwin
- Scripting tools: Perl 5.22.1
- Other open-source tools: GCC 6.3.1

TBSA-v8M test suite is distributed under Apache v2.0 license.

1.5.3 Environment requirements

The following are the minimum memory requirements for TBSA-v8M test suite to run.

- 64KB of Secure memory and 16KB of Non-secure memory to store VAL or PAL functions and related data
- 16KB of Secure Contiguous Memory for secure tests
- 16KB of Non-secure Contiguous Memory for Non-secure tests
- 4KB of NSC memory

This memory must be usable for both code execution and data access.

Note

- There is no specific requirement on the number of MPU and SAU regions for TBSA-v8M test suites.
 - For Non-secure and NSC memory, it is assumed that SAU or IDAU regions are programmed correctly with Non-secure and Non-secure Callable attributes respectively before getting into `tbsa_entry` point.
-

To learn the details of your hardware environment, the validation tests read the `tbsa_tgt.cfg` file. Refer to [Chapter 2 Porting steps on page 2-18](#).

1.6 Feedback, contributions, and support

For feedback, use the GitHub Issue Tracker that is associated with this repository.

For support, send an e-mail to support-psa-arch-tests@arm.com with the details.

Arm licensees can contact Arm directly through their partner managers. Arm welcomes code contributions through GitHub pull requests. See GitHub documentation on how to raise pull requests.

Chapter 2

Porting steps

Read this chapter for information on configuring the test suite.

It contains the following section:

- [2.1 Prerequisites for running DPM related tests in TBSA-v8M test suite](#) on page 2-19.

2.1 Prerequisites for running DPM related tests in TBSA-v8M test suite

Since DPM scenarios in TBSA-v8M involve verifying whether debug access is allowed in a target or not based on DPM settings, an external debugger is required.

The handshaking and data transfer protocol between the external debugger and Processing Element is handled by two memory mapped registers named as data and flag register. The description of these registers is given in the following table.

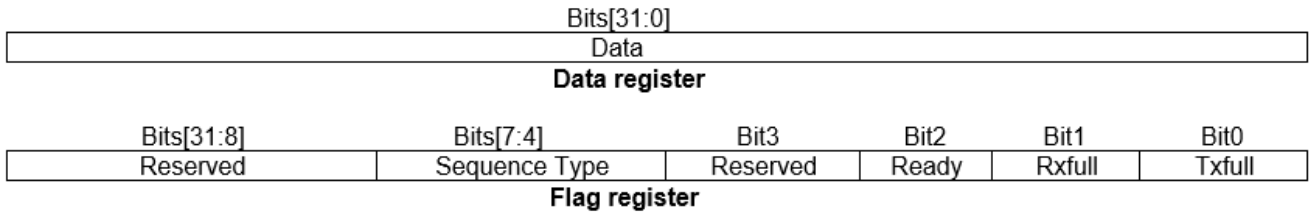


Figure 2-1 Data and flag registers

Table 2-1 Bit field and description

Bit field	Description
Txfull	This bit is set when CPU writes to Data register. This bit is cleared when debugger reads the data from Data register.
Rxfull	This bit is set when debugger writes to the data register and cleared when CPU reads the Data register
Ready	When this bit is set, debugger is ready to receive commands (Data).
Sequence Type	This field indicates debugger to choose appropriate sequence, which the test expects.

Note

- The memory mapped address for these two registers must be chosen in such a way that debug access to these addresses are allowed even when Invasive Debug is not allowed. The address for these two registers should be populated in `syscomp_tbsa_m/boards/<platform_name>/tbsa_tgt.cfg`.
- A reference program for DS-5 external debugger is available at `syscomp_tbsa_m/tools/debug/debugger_script.py`. If you use your specific debugger, then you must port this program to use in your debugger-specific commands.

Chapter 3

Architecture test suite

Read this chapter for information on the tests that are provided with the test suite.

It contains the following sections:

- [3.1 Test layering details on page 3-21.](#)
- [3.2 Build flow on page 3-22.](#)
- [3.3 Test execution flow on page 3-23.](#)
- [3.4 Test dispatcher on page 3-25.](#)
- [3.5 Test naming conventions on page 3-27.](#)
- [3.6 Test status reporting on page 3-28.](#)

3.1 Test layering details

TBSA-v8M tests are self-checking, portable C-based tests with directed stimulus.

The tests use the layered software-stack approach to enable porting across different test platforms. The constituents of the layered stack are:

1. Test suite
2. VAL
3. PAL

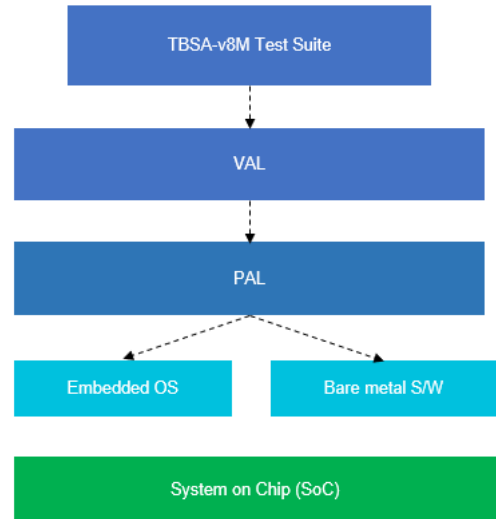


Figure 3-1 Layered software stack

The following table describes the different layers of the test.

Table 3-1 Test layers and descriptions

Layer	Description
Test suite	Collection of targeted tests that validate the compliance of the target system. These tests use interfaces that are provided by the VAL.
VAL	Contains subdirectories for the VAL libraries. This layer provides a uniform and consistent view of the available test infrastructure to the tests in the test pool. The VAL makes appropriate calls to the PAL to achieve this functionality. This layer is not ported when the underlying hardware changes.
PAL	This layer is the closest to hardware and is aware of underlying hardware details. Since it interacts with hardware, it must be ported or tailored to specific hardware required for system components present in a platform. It is also responsible for presenting a consistent interface to the validation abstraction layer required for the tests.

3.2 Build flow

Each test file must have a metadata section that describes the high-level details of the test such as Test ID, test name, and secure specification requirement. The metadata is filled in the form of macros.

As a precompile step, the metadata in each file is extracted, parsed, and saved in a separate file. The metadata file is used during the package creation phase to fill the `tbsa_test_hdr`. VAL and PAL files are not required to have a metadata section.

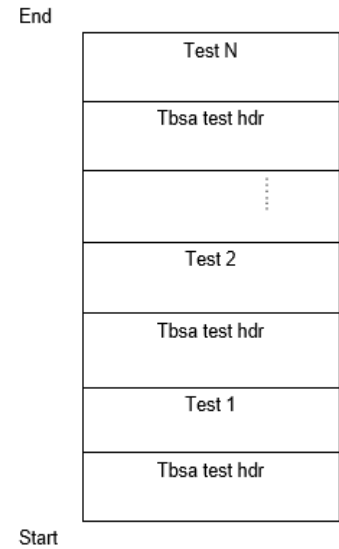


Figure 3-2 Build flow

Build steps

To compile and build the test images for the complete test suite, use the following command:

```
make TARGET=<target_directory_name>
```

For example,

```
make TARGET=fvp
```

Build outputs

The output of the build process is a collection of tests along with metadata of each test packaged as a single binary file and an ELF file. These files are:

- `tbsa_test_combined.bin`
- `tbsa.elf`

Both `tbsa_test_combined.bin` and `tbsa.elf` are available under `./out/<target_directory_name>/`

The following figure shows the directory structure after building the test ELF images.

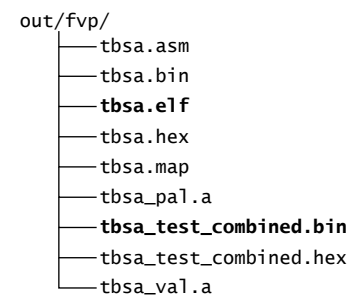


Figure 3-3 TBSA test suite directory structure

3.3 Test execution flow

The *System Under Test* (SUT) boots to an environment that enables the PAL functionality.

At this point, the SUT boot software gives control to the TBSA-v8M test suite application entry point `tbsa_entry` that comes from ELF header of `tbsa.elf` image in Secure privileged mode.

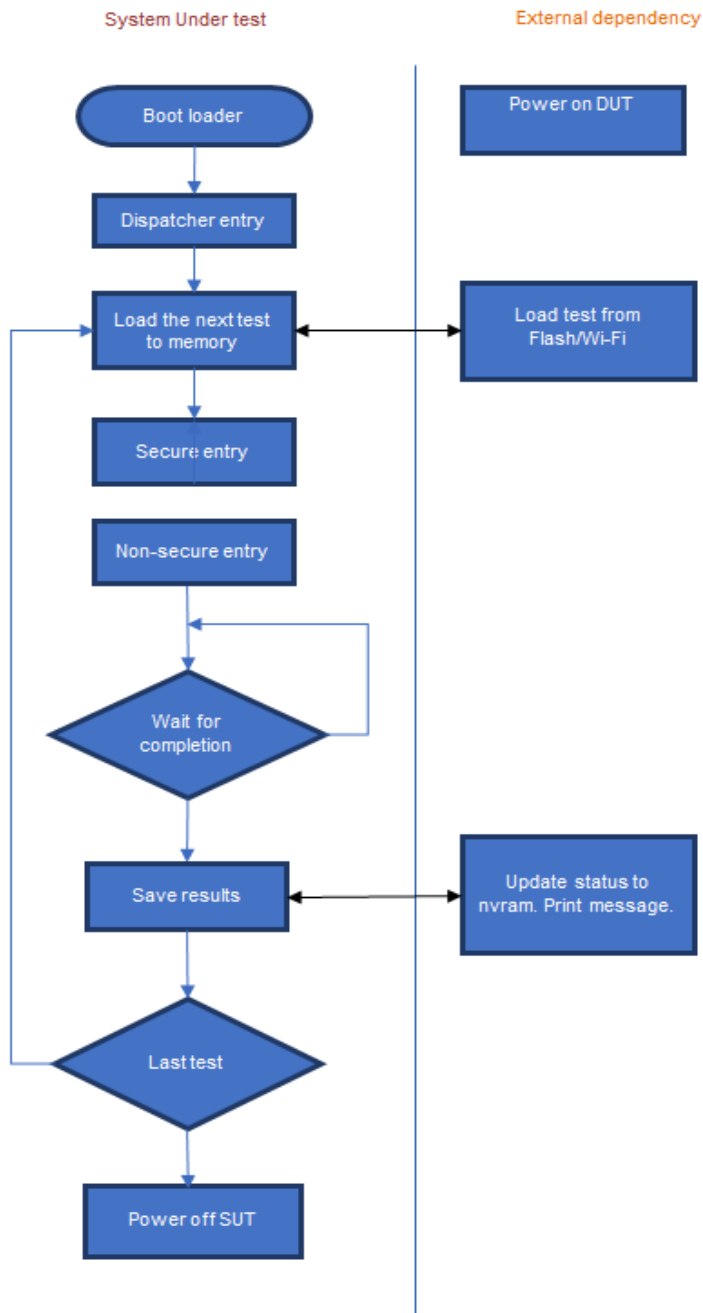


Figure 3-4 Test execution flow

The TBSA-v8M test suite application queries the VAL layer to get the necessary information to run the tests. This information includes memory maps, interrupt maps, and hardware controller maps.

Due to RAM and flash size constraints, all the tests may not be available at the same time. The dispatcher in the VAL queries the PAL to load the next test on the completion of the present test. The

PAL may optionally communicate with the external world to load the next test. Also, the dispatcher makes VAL (and in turn PAL) calls to save and report each of the test results.

Each test must present Secure and Non-secure entry points. If a scenario does not warrant either Secure or Non-secure functionality, then the entry functions will be empty.

To achieve some of the above-mentioned functions, the PAL may optionally make calls that are handled outside the system under test.

The following are some of the responsibilities that require external support:

- Feed the tests to the design under test.
- Collate and print the test status and messages.

Power ON/OFF the system as required by the test sequence. The environment in which a host test harness is running is beyond the scope of this document. But, it is envisioned that the SUT is communicating with the host using Serial port, JTAG, Wi-Fi, USB or any other means that allow for access to the external world.

3.4 Test dispatcher

Each test must present the following test entry points to the dispatcher.

1. Secure
 - a. Entry hook
 - b. Payload
 - c. Exit hook
2. Non-secure
 - a. Entry hook
 - b. Payload
 - c. Exit hook

To each of the entry points, the dispatcher passes a pointer to a structure containing the function pointers to all the available VAL functions. The Secure entry points receive the function pointers to the Secure VAL APIs. The Non-secure entry points receive function points to the Non-secure wrapper functions in the NSC region. These functions make the appropriate secure VAL function call.

The dispatcher first makes the Secure function calls and on success, calls the Non-secure functions.

The flow of the dispatcher is described in the following steps.

1. Request VAL to load the metadata of the next test into the main memory.
2. Parse the metadata that is associated with the test.
3. Verify that test is compatible with the system under test.
4. Load the test code and data sections to the appropriate locations in the main memory.
5. Call the Secure entry hook function of the test.
6. Call the Secure payload test function.
7. Call the Secure exit hook.
8. Call the Non-secure entry hook function.
9. Call the Non-secure payload function.
10. Call the Non-secure exit hook function.
11. Signal test completion and log test status.

The results from execution of each test are saved to memory. If a display console is not available, the PAL must make available the test results to the external world through other means.

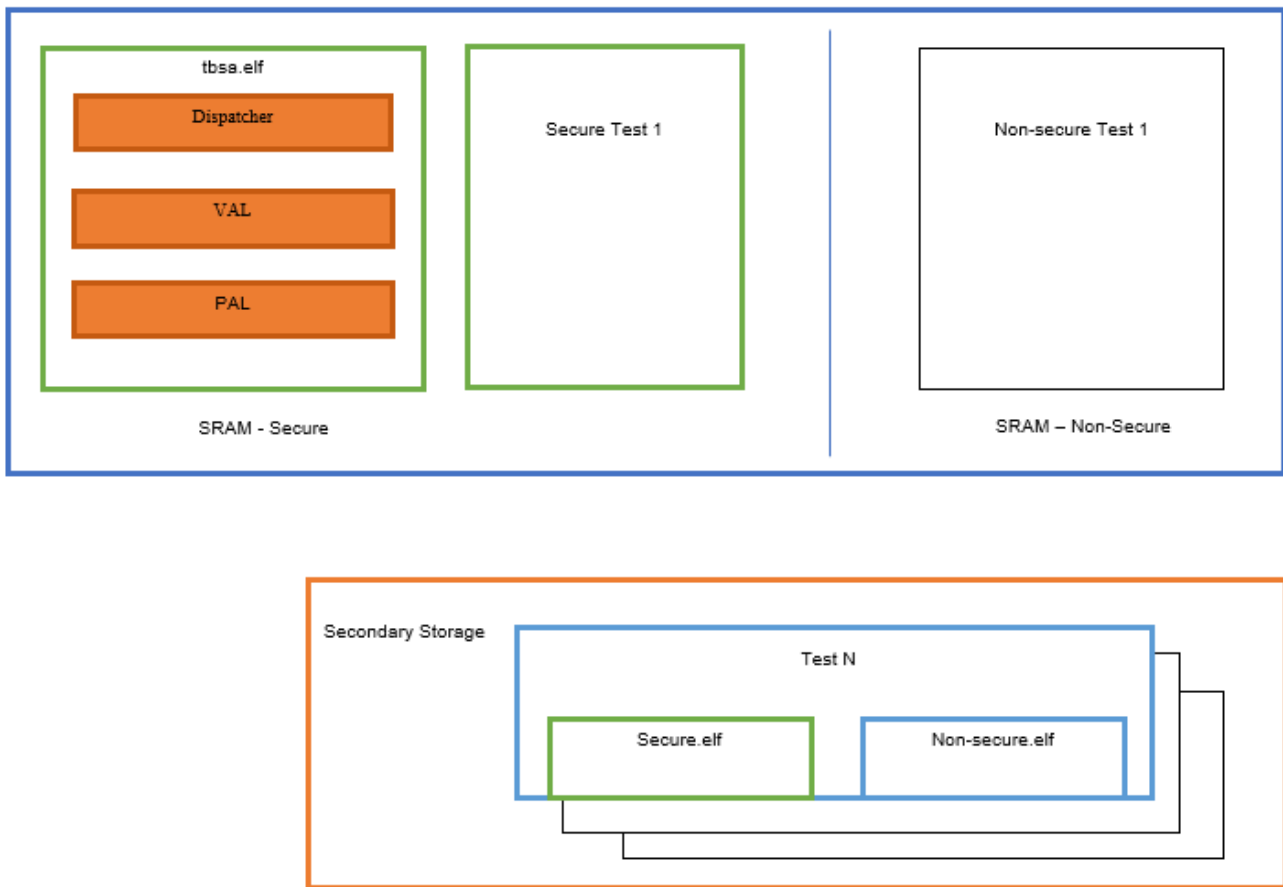


Figure 3-5 Loading a test from secondary storage

The `tbsa.elf` is loaded by the bootloader or embedded OS to the Secure memory. The dispatcher within the `tbsa.elf` loads each of the Secure test section to Secure memory and Non-secure test section to Non-secure memory. The addresses of the various sections are predetermined for a given platform as defined by the user and vary for different targets.

3.5 Test naming conventions

TBSA-v8M tests are named based on the components as defined in the following table.

Table 3-2 Test naming conventions

Component name	Base address	Test number range
Base	0	1-19
Boot	1	21-39
Crypto	2	41-59
Debug	3	61-79
EIP	4	81-99
Interrupts	5	101-119
Secure RAM	6	121-139
Peripherals	7	141-159
Trusted timers	8	161-179
Version counters	9	181-199

Each component can have a maximum of 19 tests. Hence the test numbers will be incremented from their respective base values.

For example, if you see a display as shown below, then for the Secure RAM, the component base value is 200. The test name reports it as 201, which means that it is the first test in the Secure RAM component.

```
----- Secure RAM Tests -----
201 : "R180_TBSA_INFRA"
```

3.6 Test status reporting

When the test suite is run on a target platform, each successfully run test of the suite must report either PASS or SKIP.

The following is a snippet of a successful pass test on the display console.

```

----- Secure RAM Tests -----
201 : "R180_TBSA_INFRA"
      "Secure RAM access from Trusted world only"
      # Secure                                PASS
      # Non Secure                            PASS

```

The following is a snippet of a successful skip test on the display console.

```

----- Secure RAM Tests -----
201 : "R180_TBSA_INFRA"
      "Secure RAM access from Trusted world only"
      # Secure                                SKIP
      # Non Secure                            SKIP

```

This section contains the following subsection:

- [3.6.1 Debugging of a failing test on page 3-28.](#)

3.6.1 Debugging of a failing test

Since each test is organized with a logical set of self-checking code, if there is a failure then searching for the relevant self-checking point will be a useful point to start off with debugging.

Consider the below snippet of a failing test on the display console.

```

----- Trusted Boot Tests -----
11  : "R020/R090_TBSA_BOOT"
      "Trusted boot operation from Trusted and Non-trusted world"
      # Secure
      Checkpoint C01 : Status = 8C
                                     FAIL
      # Non Secure
                                     FAIL

```

Here are some debugging points to consider.

1. This test is failing for 'Trusted Boot' component. Hence the test should be under `test_pool/boot/` directory.
2. The test ID is 11 which means first test in 'Trusted Boot' component. Hence the test is `test_pool/boot/test_s001` directory.
3. Each test will have a Secure portion and a Non-secure portion. In the above snippet, the failure is from the Secure portion. Hence it is required to see `test_pool/boot/test_s001/secure.c` file.
4. Since the failure is shown as 'Checkpoint C01', the first checkpoint is failing with a status '8C'.
5. Status of the failure is mapped with a structure `tbsa_status_t` that is available at `val/include/val_common.h`. In this example, '8C' means that the test is failing for an incorrect value.

Appendix A

Revisions

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

It contains the following section:

- [A.1 Revisions on page Appx-A-30.](#)

A.1 Revisions

Table A-1 Issue PJDOC-2042731200-3327

Change	Location	Affects
This is the first revision of the document.	-	All revisions

Table A-2 Differences between Issue PJDOC-2042731200-3327 and Issue 101308-01

Change	Location	Affects
Updated directory structure.	See 1.5.1 Directory structure on page 1-15 .	All revisions
Updated porting steps.	See Porting steps to create a new target .	All revisions
Updated test naming conventions.	See 3.5 Test naming conventions on page 3-27 .	All revisions

Table A-3 Differences between Issue 101308-01 and Issue 101308-02

Change	Location	Affects
Updated test naming conventions table.	See 3.5 Test naming conventions on page 3-27	All revisions.
Updated PAL APIs.	See PAL APIs	All revisions.