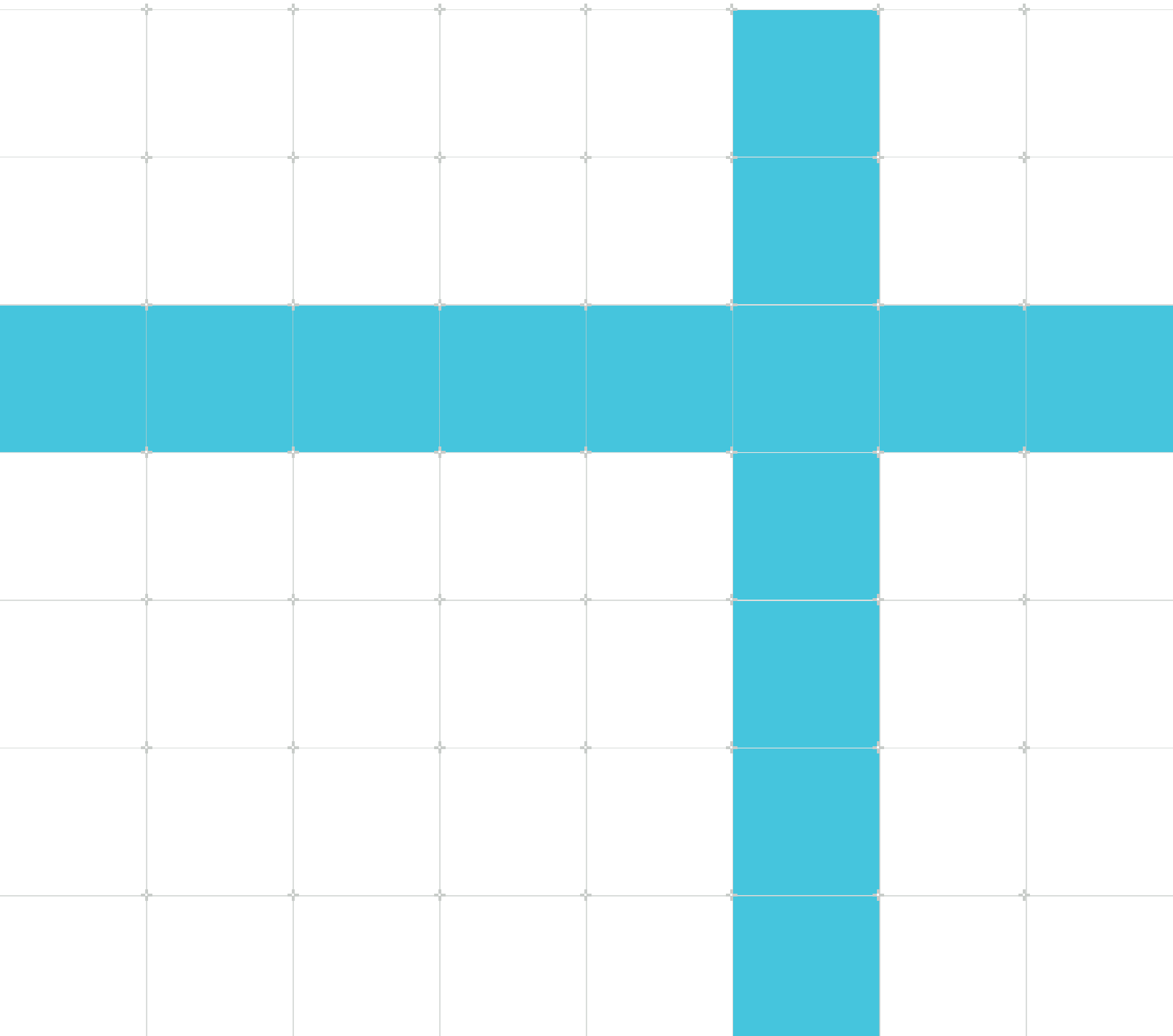
Arm® CCA RMM v1.0 Architecture Compliance Suite

Version 1.0

Validation Methodology

Issue 01
107985 _0100_01_en



Arm® CCA RMM v1.0 Architecture Compliance Suite

Validation Methodology

Release Information

Document history

Issue	Date	Confidentiality	Change
0007-01	31 March 2023	Non-Confidential	First release for v0.7
0008-01	26 September 2023	Non-Confidential	Second release for v0.8
0100-01	11 December 2023	Non-Confidential	EAC release for RMMv1.0

Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

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. Introduction

1.1 Conventions

The following subsections describe conventions used in Arm documents.

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: developer.arm.com/glossary.

Convention	Use
italic	Citations.
bold	Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<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	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

Recommendations. Not following these recommendations might lead to system failure or damage.

Requirements for the system. Not following these requirements might result in system failure or damage.

Requirements for the system. Not following these requirements will result in system failure or damage.

An important piece of information that needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

A reminder of something important that relates to the information you are reading.

1.2 Other information

See the Arm® website for other relevant information.

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

1.3 Useful resources

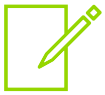
This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm® Firmware Framework for Armv9-A	DEN0077A	Non-Confidential
Arm® SMC Calling Convention	ARM DEN 0028D	Non-Confidential
Arm® Architecture Reference Manual Armv9, for Armv9-A architecture profile	DDI0487	Non-Confidential
Arm® Power State Coordination Interface Platform Design Document	DEN0022D.b	Non-Confidential
Arm® Realm Management Monitor specification	DEN0137	Non-Confidential

Arm architecture and specifications	Document ID	Confidentiality
Arm® Generic Interrupt Controller Architecture Specification GIC architecture version 3 and version 4	ID020922	Non-Confidential



Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at http://www.adobe.com.

2. Overview to CCA RMM ACS

This chapter introduces the features and components of Architecture Compliance Suite for Arm Realm Management Monitor (RMM).

2.1 Abbreviations

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

Table 2-1: Abbreviations and expansions

Abbreviation	Expansion
ABI	Application Binary Interface
ACS	Architecture Compliance Suite
API	Application Programming Interface
BSS	Block Started by Symbol
CCA	Confidential Compute Architecture
CPU	Central Processing Unit
EL	Exception Level
GOT	Global Offset Table
MMU	Memory Management Unit
MP	Multi-Processor
NS	Non-Secure
OSPM	Operating System Power Management
PAM	Platform Abstraction Layer
PSCI	Power State Coordination Interface
RME	Realm Management Extension
RMI	Realm Management Interface
RMM	Realm Management Monitor
RSI	Realm Service Interface
SMC	Secure Monitor Call
SP	Secure Partition
SPM	Secure Partition Manager
SPMC	SPM Core
SPMD	SPM Dispatcher
SUT	System Under Test
UP	Uni-Processor
VAL	Validation Abstraction Layer
VM	Virtual Machine

Abbreviation	Expansion
VMSA	Virtual Memory System Architecture
VPE	Virtual Processing Element

2.2 Realm Management Monitor

The Realm Management Monitor (RMM) is a software component that forms part of a system that implements the Arm Confidential Compute Architecture (Arm CCA). RMM is also responsible for the management of realm VMs and their interaction with the hypervisor in the normal world.

The RMM has following responsibilities in the Arm CCA system:

- Providing the services to the host.
- Allowing the host to manage the realms.

RMM also supplies services directly to the realms. The host services can be split into areas of policy and mechanics. For the policy functionality, the host owns the following policy decisions:

- When to create or destroy a realm.
- When to add or remove memory from a realm.
- When to schedule a realm.

The RMM supports the host policies by:

- Providing services to manipulate realm page tables - used in the creation or destruction and the addition or removal of realm memory.
- Managing realm context - save and restore used in scheduling interrupt support.
- PSCI call interception - there are power management requests. The RMM also provides services to realms, primarily attestation and cryptographic services.

The RMM also upholds the following security primitives for realms:

- Validates the host's requests for correctness.
- Isolates realms from each other.
- Helps in calculating the realm and platform attestation and measurements.

The RMM specification defines two communication channels to allow all functionalities to be requested and controlled between the Normal world Host and the realm VM.

2.2.1 Realm Service Interface (RSI)

A second channel defined between the RMM and the realm VM is called the RSI. The RSI is the channel for requesting services from the RMM. The RSI provides a channel for external services that allows few realm management operations to pass to the realm VM from the RMM. These

services can include cryptographic services and attestation. The RSI is also the channel for memory management requests from the realm VM to the RMM.

For more information on RMM, see the Realm Management Monitor specification.

2.2.2 Realm Management Interface (RMI)

The communication channel from the host to the RMM is called the RMI. The RMI allows the hypervisor to provide instructions to the RMM that manages the realm. The RMI uses SMC calls from the host hypervisor to request management control from the RMM. The RMI enables control of the realm management which includes the creation, population, execution, and destruction of the realms.

2.3 Architecture Compliance Suite

RMM Architecture Compliance Suite (ACS) contains a set of functional tests, demonstrating the invariant behaviors that are specified in the architecture specification. ACS tests are self-checking portable C and assembly-based tests with directed stimulus. The tests are open source and available with a BSD-3 license, allowing external contribution.

These tests are used to ensure the architecture compliance of the implementations to Arm RMM Architecture and they are not a substitute for design verification. ACS is expected to run on RMM and EL3 firmware stack on RME-enabled platforms.

For more information on tests, test steps, and test scenarios that are in scope and out of scope for compliance, see docs/testcase_checklist.md.

2.4 Directory structure

The following table describes the ACS components available in the top-level directory of the test suite when the release package is downloaded from GitHub:

Figure 2-1: Directory structure of CCA RMM test suite

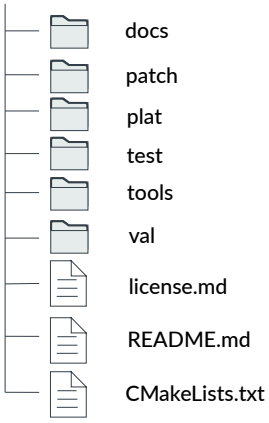


Table 2-2: Components and descriptions

Component	Description
docs	Contains the ACS documentation. Consists of Porting Guide, Test scenario document, and Validation Methodology document.
plat	Contains files to form the PAL.
test	Contains self-checking C tests.
tools	Contains CMake files used to generate test binaries and run scripts to run tests on the reference platform.
val	Contains files to form the VAL.
LICENSE.md	License information
README.md	Contains information on Arm CCA RMM ACS.
CMakeLists.txt	Contains information on the CMake build support.

3. Validation Methodology

This chapter describes the validation methodology for the Architecture Compliance Suite.

3.1 ACS software stack

This chapter describes the validation methodology for the Architecture Compliance Suite.

The ACS uses a layered software stack approach to enable porting across different test platforms. The constituents of the layered stack are:

- **Tests**

  This layer is a collection of targeted tests. It verifies if the target behavior conforms to the RMM specification. These tests map to one or more scenarios identified in the scenario document. They use interfaces that are provided by the VAL. A test does not directly interface with PAL functions. The test layer does not need any code modifications when porting from one platform to another.

- **VAL**

  This layer contains sub directories for the VAL libraries. It provides a uniform and consistent view of the available test infrastructure to the tests in the test pool. The VAL wraps RMI and RSI ABIs to exercise them in a generic way from the tests. The VAL also includes the PE initialization/boot code needed to run tests on the Host, Realm, and secure side. Any test dependency on platform information or functionality goes through via VAL functions.

- **PAL**

  This layer is the closest to the target and it must be aware of the required information about underlying hardware and firmware. Examples of platform-specific code include interactions with IO like UART. PAL must be ported or tailored to the platform on which ACS is run. For more information on porting setups refer, docs/porting_guide.md.

3.2 Design under test

The following figure describes the design under test.

Figure 3-1: Design under test

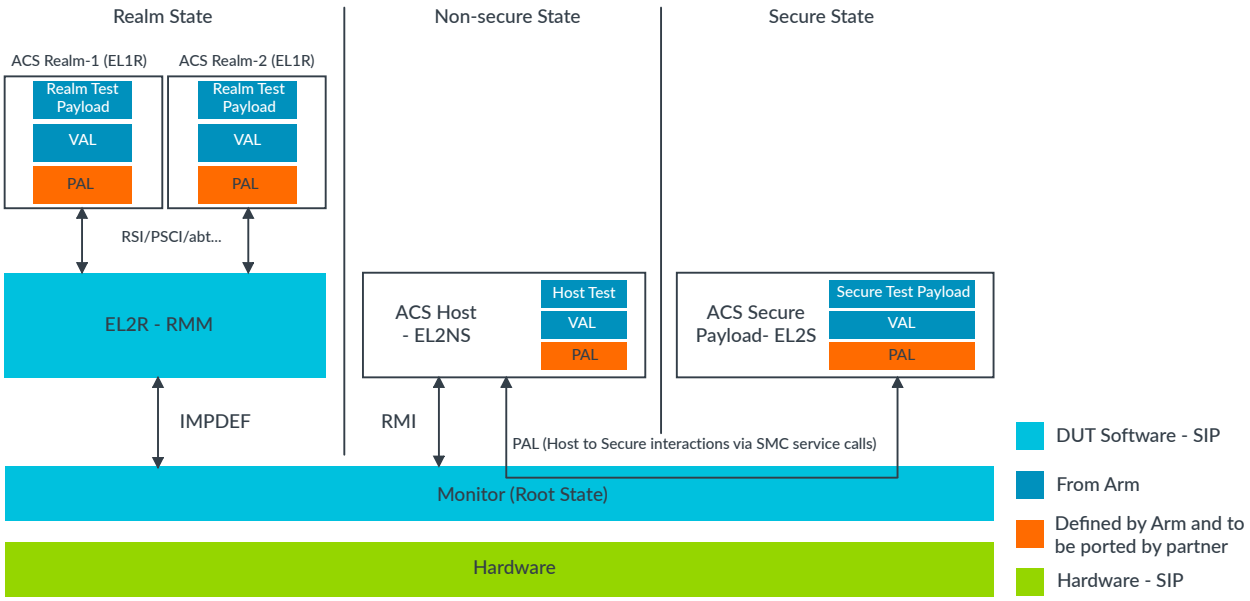


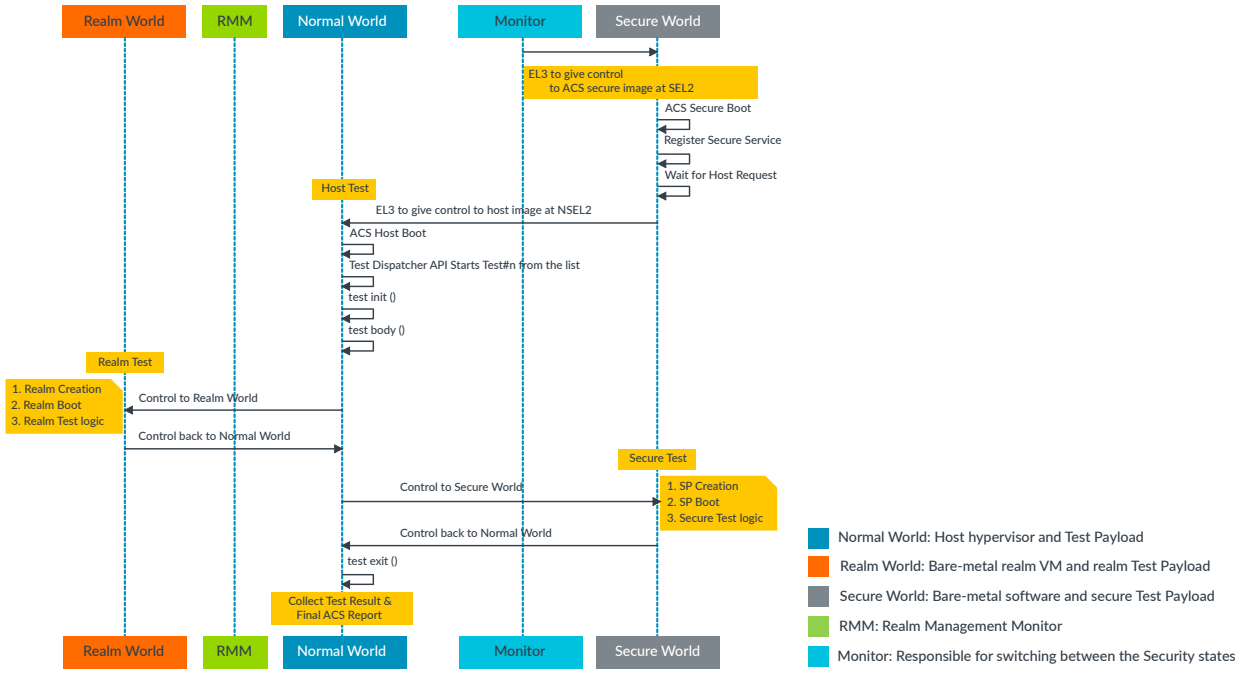
Table 3-1: Components and descriptions

Component	Description
Hardware	Hardware system supporting Realm Management Extension (RME).
Monitor	The most privileged software component is responsible for switching between the security states. The monitor runs in the root state (EL3).
RMM	The software component is responsible for the management of Realms. RMM runs in the realm world at EL2.
ACS Host	ACS host is the bare-metal hypervisor containing the ACS host test payload. It is the anchor of the test. ACS host runs at EL2NS.
ACS Realm	ACS realm is the bare-metal realm VM containing the ACS realm test payload and it runs at EL1R. At runtime, the ACS host creates a realm VM by performing the necessary set of RMI commands. Interaction between the ACS host and realm code is through RMM-defined interfaces such as RMI calls, PSCI calls, and realm exit events.
ACS Secure	ACS secure is the bare-metal software containing ACS secure test payload and it can run at EL2S as SPMC. Interaction between the ACS host and secure code is through FF-A direct messaging interfaces over the PAL layer.

3.3 Test execution flow

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

Figure 3-2: Test execution flow



The process that is followed for the flow control is:

1. System software component, like the Monitor loads ACS secure image (acs_secure.bin) in the secure memory.
2. Monitor gives control to the start of the image at EL2S.
3. ACS Secure code performs the necessary boot sequence.
4. ACS Secure code registers the service with Monitor and waits for the ACS host requests by giving control back to Monitor.
5. Monitor loads the combined image (acs_non_secure.bin) derived from ACS Host Image and ACS Realm image.
6. Monitor gives control to start the combined image at EL2NS.
7. ACS Host (anchor of the test regression) performs the necessary boot sequence and starts the test iteration loop.
8. Test entry and exits happen in the host.
9. Depending upon the test scenario, ACS Host Test will launch the ACS Realm in the Realm world and interact with ACS Realm Test or ACS Host Test will interact with ACS Secure Test.
10. Print status during the test execution as required.
11. Loop until all the checks of the test are completed. The test will exit the check loop on detecting check failure.
12. Loop until all the tests are completed.

- Host test payload expects EL3 to complete the boot of Secure world and RMM. The order is immaterial from ACS PoV.
- With respect to the test, switching between Host and Realm is based on RMM specification.
- With respect to the test, switching between Host and Secure is through PAL. And current PAL implementation is based on FF-A Direct Messaging ABIs.

3.4 ACS boot flow

ACS has common boot flow sequences for Secure, Host, and Realm with each of them having a different entry point. The below diagram shows the boot sequence for the Host.

Figure 3-3: UP Realm with single or multi thread

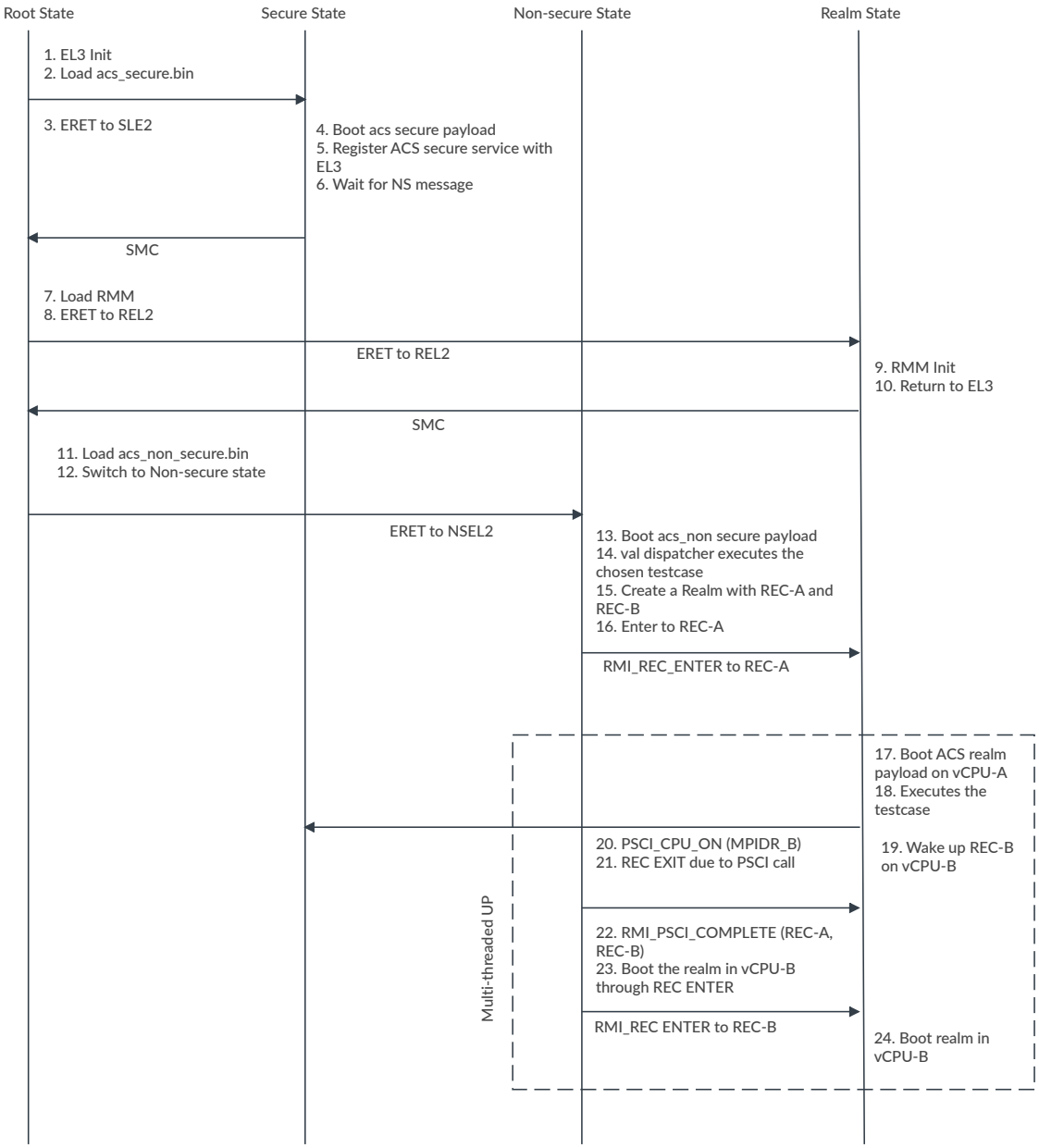
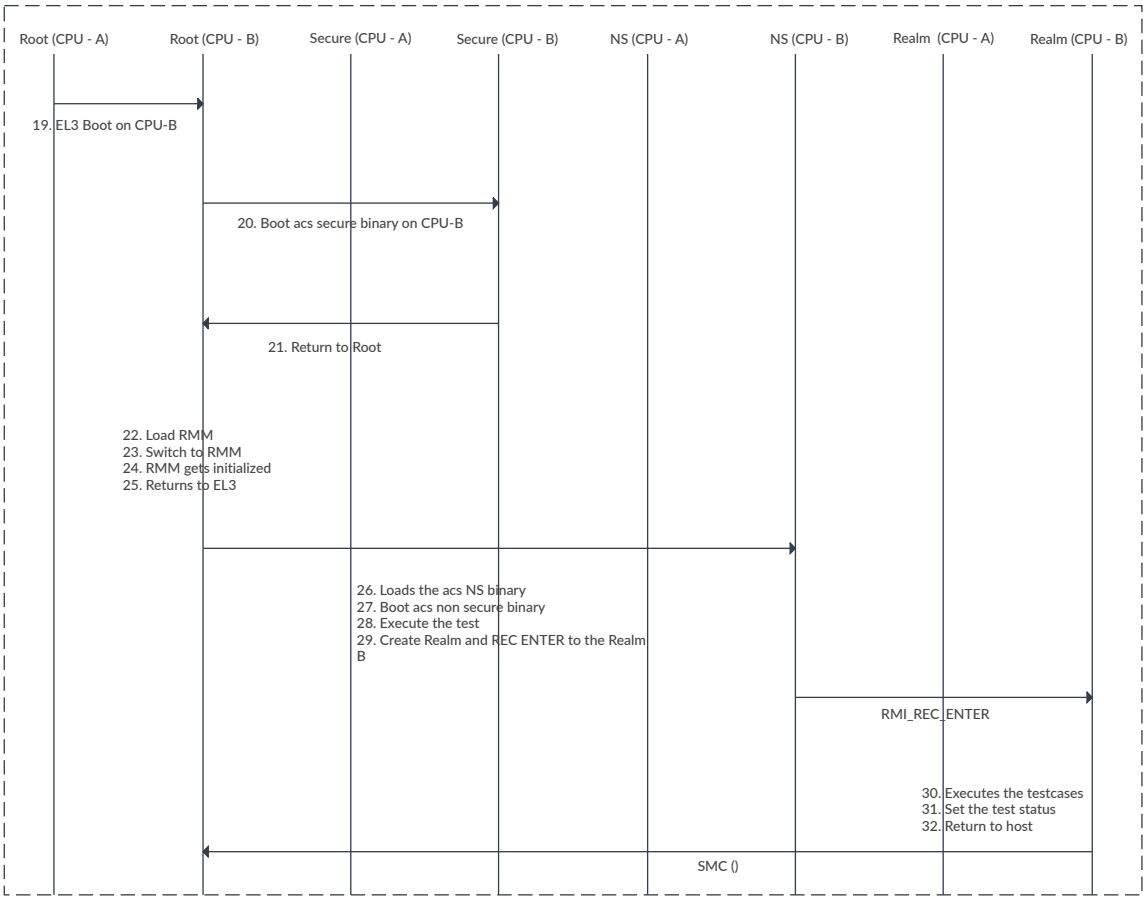


Figure 3-4: True MP Realm



The following steps explain the boot flow sequence for the Host:

1. Program the system registers to set the vector table and enable the I-cache.
2. Invalidate the data cache and instruction cache for image regions to avoid the use of any stale cache data from previous boot stages.
3. Program the stack to enable C Programs.
4. Detect the Primary vs Secondary boot and save the primary CPU ID as leading CPU.

Primary boot:

1. Clear the BSS region.
2. Write the page table for Image regions, Device regions (Ex: UART), and NS Memory Pool.
3. Enable the MMU.
4. The host starts the test iteration loop.

Secondary boot:

1. Enable the MMU.
2. The host starts the test iteration loop.

For the Realm and Secure world, the implementation of the test dispatcher performs the following:

- It executes the dedicated Realm Test payload function for the test number shared by host.
- It executes the dedicated Secure Test payload function for the test number shared by host.
- The boot flow fixes the image symbols from the source view to the destination view before executing the BSS region init sequence.

3.5 Page table setup

ACS creates a page table for Image regions, Device regions (Ex, UART), and Memory Pool at runtime as part of the boot sequence.

- The size of image regions such as CODE, DATA, and BSS regions is calculated using symbols specified in the linker script.
- The size of Device regions is derived through the PAL.
- The size of the NS memory pool region is derived through PAL. NS memory pool is divided into two regions:
    - NS Heap Memory for Host use only. Host provisions memory resources such as RD, REC, RTT, and Data granules, to RMM and Test Realm out of this range.
    - NS region shared across Secure, Realm, and the Normal world. This region is used by VAL and Test to share information across all three worlds.

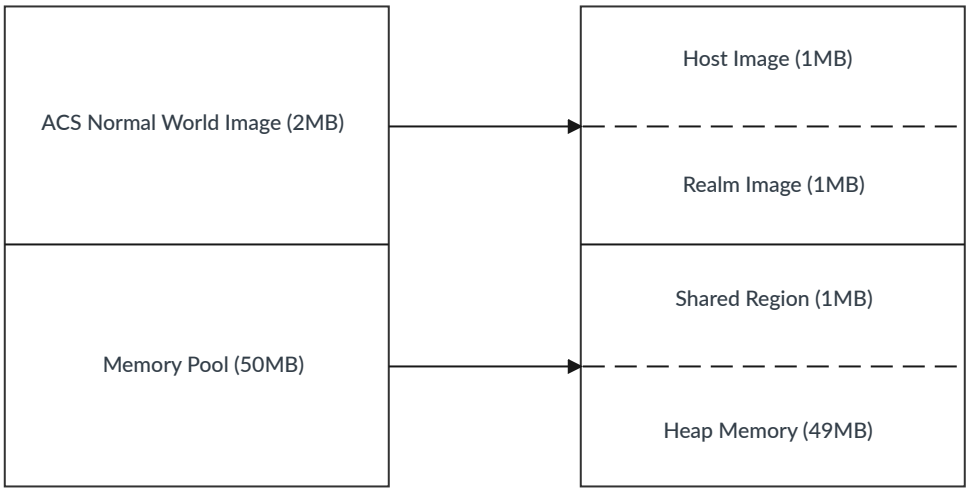
The memory for the page tables is reserved in the BSS segment based on the maximum table requirements. Boot enables the MMU with the below configuration:

- 4KB Translation Granule.
- Support for non-flat map VA to PA is available to the test writer for memory under test. rest of the regions are flat mapped.

3.6 ACS memory layout

ACS requires the following memory regions that is provided by the platform:

Figure 3-5: Memory layout



- NS Memory Pool (PAL configuration) - 52MB.
  - 2MB and 50MB regions of the NS memory pool must be contiguous but not as a whole 52MB.
  - First 2MB for loading NS host and initial Realm image. Reserve 1MB size for each image.
  - 49MB for Heap memory - Free NS range available to Host
- Host provisions below resources to RMM and Realm out of this range.
  - RDs, RTTs, RECs, Data Granules, Sharing of NS Granules to provide RD parameters, reference results, syncing threads in host and Realm, run objects, Granules for testing failure conditions.
- VAL provides dynamic allocation APIs to test to allocate memory from this pool.
  - 1MB of NS Shared Region.
- Information sharing across the world happens through this region. This region is mapped (stg1) across all worlds.
- S Memory Pool (PAL Configuration) - 1MB.
  - Memory reserve for Secure Test payload Image.

3.7 Test report and analyzing results

When the ACS is run on the platform, each successful test must report either PASS or SKIP. The following is an example code of a successful test pass.

```
*********************************

Suite=exception : Test=exception_rec_exit_wfi
                  [Host-Check  1] : REC ENTER WFI Verified
                  [Realm-Check 1] : Realm WFI Trigger checks are verified
Result => Passed

*********************************
Suite=exception : Test=exception_rec_exit_ripas
                  [Realm-Check 1] : RIPAS Value RAM --> EMPTY verified
                  [Realm-Check 2] : RIPAS RAM --> EMPTY verified with ACCEPT SET
                  [Realm-Check 3] : RIPAS Value RAM --> RAM verified with the PARTIAL
 RIPAS SET
                  [Realm-Check 4] : RIPAS Value RAM --> RAM verified with REJECT
Result => Passed

*********************************
```

The following code displays the status of all the tests scheduled in a single execution run.

```
REGRESSION REPORT:
==================
   TOTAL TESTS      :
   TOTAL PASSED     :
   TOTAL FAILED     :
   TOTAL SKIPPED    :
   TOTAL SIM ERROR  :

******* END OF ACS *******
```

If a test fails or skips, then you will see extra print messages to determine the cause.

Consider the following when debugging the failed test:

- Since each test is organized with a logical set of self-checking codes, if a failure occurs, searching for the relevant self-checking point is a useful point to start debugging.
    - Along with the error message, the test also prints the file and line number from where the error message is printed.
    - Test results contain the error code associated with the error message. The status of the error code is mapped with a structure val_status_t that is available at val/inc/val.h. Look for the enum that is dedicated to this number to see the status in the verbatim form.
- If the default prints does give enough information, you can recompile and rerun the test binaries with a high print verbosity level. See the test suite build README to understand how the test verbosity can be changed.

3.8 Recovery from EL3/REL2 fault

Based on the test scenario, the test will perform a sequence to trigger expected fault conditions at EL3 or RMM level. For example, test triggering GPC fault. In such conditions, ACS needs platform support to redirect control to ACS Host or Realm to be able to conclude the current test results and execute the next test.

The following are the error handling steps if EL3/RMM supports the injection of faults into Host/Realm (abort injection from higher Exception Level (EL) to lower EL):

- Test installs the handler at the expected vector table location. For example, install a synchronous abort handler for stage 1 Data Abort.
- The test performs the authorized access and expects fault handling at EL3 or RMM.
- EL3/RMM injects the fault at lower EL by copying ESR.EC and FAR system registers values and updates ELR_ELx with VBAR+handlet_offset.
- Host/Realm receives the abort at the installed handler which concludes the error condition and returns to the interrupted code.

3.9 MP execution setup

On power-on reset, it is expected that ACS boots only on a single physical or Virtual PE (also called the primary PE) for ACS secure and ACS Host. ACS MP execution environment follows the <Power State Coordination Interface (PSCI)> specification.

The secondary execution context setup for the Host is as follows:

- The primary boot PE code uses the PSCI_CPU_ON interface to request the system software to initialize another PE of a given MPID and execute host code from the entry address specified during the PSCI_CPU_ON call.
- Handshaking and synchronization between Primary PE and secondary PEs are based on a spin-lock mechanism
- When the task on secondary PE is over, secondary PE uses the PSCI_CPU_OFF interface to exclude itself from the system.

The secondary execution context setup for Secure is as follows:

- Power management in the Secure world is as per FF-A specification. As described in the specification, the secure world does not perform power management independently from the Normal world. Instead, it is informed about OSPM operations initiated by the Normal world through PSCI functions.
- Secure code responses to the PSCI ON/OFF invocation from the Host and initialize the secondary PE.

MP execution setup for the Realm is followed as the RMM specification.

4. Appendix

4.1 Revisions

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

Table 4-1: A.1 Issue 0007-01

Change	Location
First release	-

Table 4-2: A.2 Differences between Issue 0007-01 and Issue 0008-01

Change	Location
Updated the code for the tests scheduled in a single execution run.	See, 3.7 Test report and analyzing results on page 20.

Table 4-3: A.3 Differences between Issue 0008-01 and Issue 0100-01

Change	Location
Updated the Page table setup description.	See, 3.5 Page table setup on page 19.