



Arm[®] SBSA Architecture Compliance Bare-metal

Version 8.0.0

User Guide

Non-Confidential

Copyright © 2020–2025 Arm Limited (or its affiliates).
All rights reserved.

Issue 01

102311_0800_01_en



Arm® SBSA Architecture Compliance Bare-metal User Guide

This document is Non-Confidential.

Copyright © 2020–2025 Arm Limited (or its affiliates). All rights reserved.

This document is protected by copyright and other intellectual property rights.

Arm only permits use of this document if you have reviewed and accepted [Arm's Proprietary Notice](#) found at the end of this document.

This document (102311_0800_01_en) was issued on 2025-12-10. There might be a later issue at <https://developer.arm.com/documentation/102311>

The product version is 8.0.0.

See also: [Proprietary Notice](#) | [Product and document information](#) | [Useful resources](#)

Start reading

If you prefer, you can skip to [the start of the content](#).

Intended audience

This book is written for engineers who are designing or verifying an implementation of the Arm® Server Base System Architecture.

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.

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>.

Contents

1. Overview to SBSA ACS.....	5
1.1 Abbreviations.....	5
1.2 SBSA ACS.....	6
1.3 ACS design.....	6
1.4 Boot framework.....	7
1.4.1 Boot process and boot flow.....	7
1.4.2 Boot framework for Bare-metal.....	8
1.5 Steps to customize bare-metal code.....	9
1.5.1 Test suites.....	10
2. Execution of SBSA ACS.....	11
2.1 SoC emulation environment.....	11
2.1.1 PE.....	11
2.1.2 PCIe.....	13
2.1.3 DMA.....	15
2.1.4 SMMU and device tests.....	16
2.1.5 GIC.....	19
2.1.6 Timer.....	20
2.1.7 Watchdog timer.....	21
2.1.8 Memory.....	21
2.1.9 MPAM.....	22
2.1.10 HMAT.....	25
2.1.11 RAS.....	26
2.1.12 PMU.....	26
2.1.13 MMU Configuration.....	27
2.1.14 Peripherals.....	27
2.2 Bare-metal Boot.....	28
3. Porting requirements.....	29
3.1 PAL implementation.....	29
3.1.1 PE.....	29
3.1.2 GIC.....	30
3.1.3 Timer.....	30

3.1.4 IOVIRT.....30

3.1.5 PCIe..... 31

3.1.6 SMMU.....32

3.1.7 Peripheral..... 33

3.1.8 MPAM..... 33

3.1.9 RAS..... 34

3.1.10 DMA..... 34

3.1.11 Exerciser.....35

3.1.12 Memory map..... 36

3.1.13 Miscellaneous..... 36

4. SBSA ACS flow..... 38

4.1 SBSA ACS flow diagram..... 38

4.2 SBSA test example flow..... 39

Proprietary Notice..... 40

Product and document information.....42

Product status..... 42

Revision history.....42

Conventions.....45

Useful resources.....48

1. Overview to SBSA ACS

This chapter provides an overview on Arm SBSA ACS, the ACS design, and steps to customize the bare-metal code.

1.1 Abbreviations

The following table lists the abbreviations used in this document.

Table 1-1: Abbreviations and expansions

Abbreviation	Expansion
ACS	Architecture Compliance Suite
ETE	Embedded Trace Extension
DMA	Direct Memory Access
ECAM	Enhanced Configuration Access Mechanism
GIC	Generic Interrupt Controller
HMAT	Heterogenous Memory Attribute Table
IORT	Input Output Remapping Table
IOVIRT	Input Output Virtualization
ITS	Interrupt Translation Service
MMU	Memory Management Unit
MPAM	Memory System Resource Partitioning and Monitoring
MPIDR	Multiprocessor ID Register
MSI	Message-Signaled Interrupt
PAL	Platform Abstraction Layer
PCC	Platform Communications Channel
PCIe	Peripheral Component Interconnect Express
PE	Processing Element
PMU	Performance Monitoring Unit
PPTT	Processor Properties Topology Table
RAS	Reliability, Availability, and Serviceability
RC	Root Complex
RP	Root Port
SBSA	Server Base System Architecture
SoC	System on Chip
SMC	Secure Monitor Call
SMMU	System Memory Management Unit
SRAT	System Resource Affinity Table
UART	Universal Asynchronous Receiver and Transmitter

Abbreviation	Expansion
UEFI	Unified Extensible Firmware Interface
VAL	Validation Abstraction Layer

1.2 SBSA ACS

Arm specifies a hardware system architecture which is based on Arm 64-bit architecture that server system software such as operating systems, hypervisors, and firmware can rely on. This ensures standard system architecture to enable a suitably built single OS image to run on all the hardware compliant with this specification.

Arm provides the SBSA Architecture Compliance Suite (ACS) which contains self-checking portable C-based test cases to verify the compliance of hardware platforms to Server Base System Architecture (SBSA).

For more information on Arm SBSA ACS, see the [README](#).

1.3 ACS design

The ACS is designed in a layered architecture that consists of the following components:

- Platform Abstraction Layer (PAL) is a C-based, Arm-defined API that you can implement. It abstracts features whose implementation varies from one target system to another. Each test platform requires a PAL implementation of its own. 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.
 - For each component, PAL implementation must populate a data structure which involves supplying SoC-specific information such as base addresses, IRQ numbers, capabilities of PE, PCIe, RC, SMMU, DMA, and others.
 - PAL also uses client drivers underneath to retrieve certain device-specific information and to configure the devices.
- Validation Abstraction Layer (VAL) provides an abstraction over PAL and does not change based on the platform. This layer uses PAL layer to achieve a certain functionality. The following example achieves read memory functionality.

```
val_pcie_read_cfg -> pal_pcie_read_cfg
```
- Test pool is a layer which contains a list of test cases implemented for each component.
- Application is the top-level layer which allocates memory for component-specific tables and executes the test cases for each component.

The ACS test components are classified as follows:

- PE
- GIC

- PCIe
- Exerciser
- I/O virtualization
- SMMU
- Watchdog
- Memory
- MPAM
- PMU
- RAS
- HMAT
- NIST
- ETE

1.4 Boot framework

The bootwrapper is a simple implementation of a boot loader to boot up the system and transition to the ACS where specific set of tests are run.

The bootwrapper initializes the hardware and loads the ACS into the memory, allowing the system to start up, independent of UEFI and execute ACS tests automatically. This further reduces porting complexity for the partners and provides them with off-the-shelf system Init code.

1.4.1 Boot process and boot flow

The boot process is the sequence of operations that occurs when a computer system is powered-on or restarted, allowing it to transition from a power-off state to an operational state, where the operating system can run.

A boot loader, also known as a boot manager, is a piece of software responsible for initiating the boot process and loading the operating system into memory. Boot loader is located in firmware or a dedicated boot partition and is executed when the system is powered-on or restarted.

The cold boot path in this implementation of TF-A depends on the execution state. For AArch64, it is divided into five steps (in the order of execution):

- Boot Loader stage 1 (BL1) - AP Trusted ROM
- Boot Loader stage 2 (BL2) - Trusted Boot Firmware
- Boot Loader stage 3-1 (BL31) - EL3 Run time Software
- Boot Loader stage 3-2 (BL32) - Secure-EL1 Payload (optional)
- Boot Loader stage 3-3 (BL33) - Non-trusted Firmware

1.4.2 Boot framework for Bare-metal

With the introduction of bootwrapper, the UEFI layer is bypassed in the ACS boot flow. SBSA ACS with bootwrapper runs as non-trusted firmware at BL33.

The following figures show the overview of System boot flow and ACS boot flow in a system environment.

Figure 1-1: System boot flow

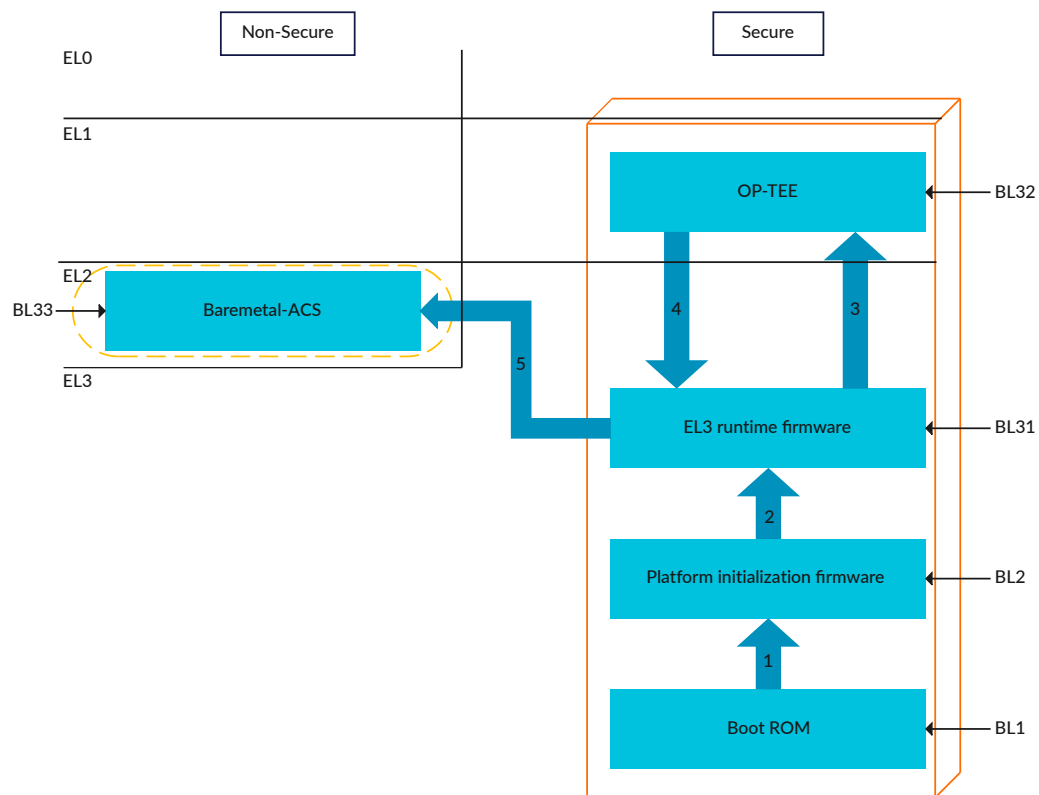
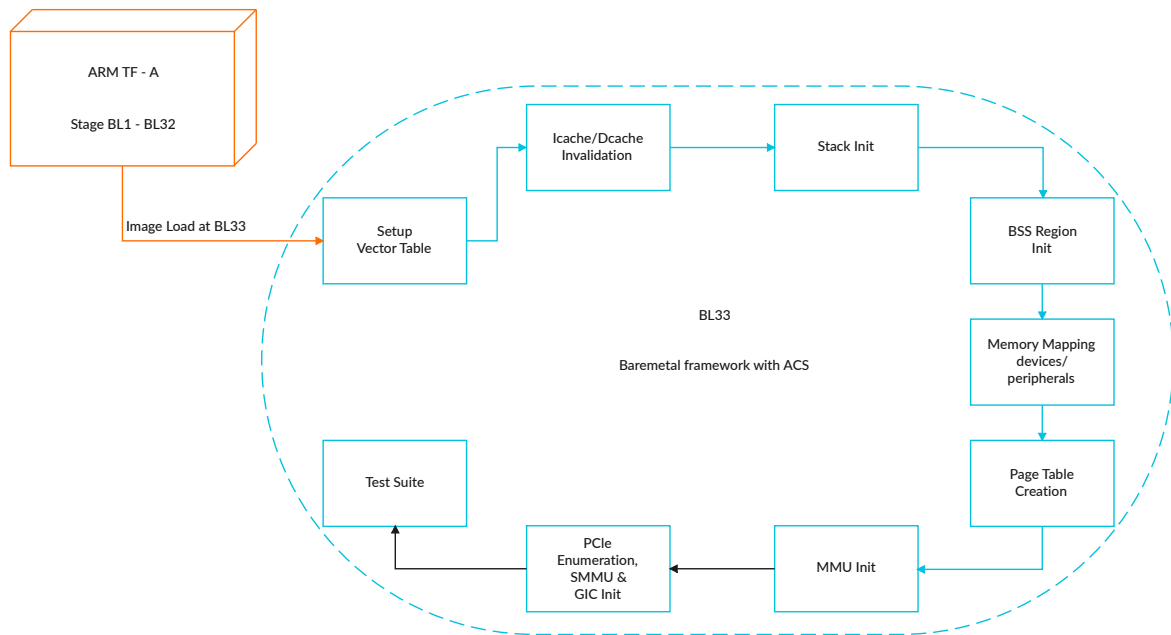


Figure 1-2: ACS Boot framework flow

1.5 Steps to customize bare-metal code

The following are the steps to customize bare-metal code for different platforms.



The `pal_baremetal` reference code is located in [baremetal](#).

1. Run the command `./generate.py <platform_name>`.

```
mkdir <platform_name>
```
2. Running the command creates the folder with the platform name and the PAL APIs under `target/<platform_name>/src/pal_sbsa.c` to be filled.

```
cp -r RDN2/ platform_name/
```
3. Port all the required APIs. For more details on the list of APIs, see the [Porting requirements](#).
4. Modify the files `target/<platform_name>/include/platform_override_fvp.h` and `target/<platform_name>/include/platform_image_def.h` with platform-specific information. For more details on sample implementation, see the [Execution of SBSA ACS](#).

1.5.1 Test suites

The following table lists the bare-metal components for each test implementation.

Table 1-2: Bare-metal components

Test suites	Files
PE	pal_pe.c
GIC	pal_gic.c
PCIe	pal_pcie.c, pal_pcie_enumeration.c
Exerciser	pal_exerciser.c
IOVIRT	pal_iovirt.c
SMMU	pal_smmu.c
Timer and Watchdog	pal_timer_wd.c
Peripherals (UART and Memory)	pal_peripherals.c
DMA	pal_dma.c
MPAM	pal_mpam.c
PMU	pal_pmu.c
RAS	pal_ras.c
HMAT	pal_hmat.c
Miscellaneous	pal_misc.c



Note

PAL implementation requires porting when the underlying platform design changes.

2. Execution of SBSA ACS

This chapter provides information on the execution of the SBSA ACS on a full-chip SoC emulation environment.

2.1 SoC emulation environment

Executing SBSA ACS on a full-chip emulation environment requires implementation of PAL. This involves providing a collection of SoC-specific information such as capabilities, base addresses, IRQ numbers to the test logic.

In Unified Extensible Firmware Interface (UEFI) base systems, all the static information is present in UEFI tables. The PAL implementation which is based on UEFI, uses the generated header file for populating data structures. For a bare-metal system, this information must be supplied in a tabular format which becomes easy for PAL API implementation.

2.1.1 PE

This section provides information on the number of PEs in the system.

PE-specific information

Tests contain comparison of Multiprocessor ID Register (MPIDR) values with actual values read from register. Such interrupts are generated for the Performance Monitoring Unit (PMU) lines and tested.

PLATFORM_OVERRIDE_PEx_MPIDR:

MPIDR register value represents the xth PE hierarchy (cluster, core).

PLATFORM_OVERRIDE_PEx_INDEX:

Represents the xth PE.

PLATFORM_OVERRIDE_PEx_PMU_GSIV:

PMU interrupt number for xth PE.

A platform with eight PEs is populated as follows:

```
#define PLATFORM_OVERRIDE_PE_CNT      0x8
#define PLATFORM_OVERRIDE_PE0_INDEX  0x0
#define PLATFORM_OVERRIDE_PE0_MPIDR  0x0
#define PLATFORM_OVERRIDE_PE0_PMU_GSIV 0x17

#define PLATFORM_OVERRIDE_PE1_INDEX  0x1
#define PLATFORM_OVERRIDE_PE1_MPIDR  0x100
#define PLATFORM_OVERRIDE_PE1_PMU_GSIV 0x17

#define PLATFORM_OVERRIDE_PE2_INDEX  0x2
#define PLATFORM_OVERRIDE_PE2_MPIDR  0x200
```

```
#define PLATFORM_OVERRIDE_PE2_PMU_GSIV 0x17

#define PLATFORM_OVERRIDE_PE3_INDEX      0x3
#define PLATFORM_OVERRIDE_PE3_MPIDR     0x300
#define PLATFORM_OVERRIDE_PE3_PMU_GSIV 0x17

#define PLATFORM_OVERRIDE_PE4_INDEX      0x4
#define PLATFORM_OVERRIDE_PE4_MPIDR     0x10000
#define PLATFORM_OVERRIDE_PE4_PMU_GSIV 0x17

#define PLATFORM_OVERRIDE_PE5_INDEX      0x5
#define PLATFORM_OVERRIDE_PE5_MPIDR     0x10100
#define PLATFORM_OVERRIDE_PE5_PMU_GSIV 0x17

#define PLATFORM_OVERRIDE_PE6_INDEX      0x6
#define PLATFORM_OVERRIDE_PE6_MPIDR     0x10200
#define PLATFORM_OVERRIDE_PE6_PMU_GSIV 0x17

#define PLATFORM_OVERRIDE_PE7_INDEX      0x7
#define PLATFORM_OVERRIDE_PE7_MPIDR     0x10300
#define PLATFORM_OVERRIDE_PE7_PMU_GSIV 0x17
```

Header file representation:

```
typedef struct {
    uint32_t num_of_pe;
} PE_INFO_HDR;

/**
 * @brief structure instance for PE entry
 */

typedef struct {
    uint32_t pe_num; /* PE Index */
    uint32_t attr; /* PE attributes */
    uint64_t mpidr; /* PE MPIDR */
    uint32_t pmu_gsiv; /* PMU Interrupt */
    uint32_t gmain_gsiv; /* GIC Maintenance Interrupt */
    uint32_t acpi_proc_uid; /* ACPI Processor UID */
    uint32_t level_1_res[MAX_L1_CACHE_RES]; /* index of level 1 cache(s) in
    cache info table */
    uint32_t trbe_interrupt; /* TRBE Interrupt */
} PE_INFO_ENTRY;

typedef struct {
    PE_INFO_HDR header;
    PE_INFO_ENTRY pe_info[];
} PE_INFO_TABLE;
```

SMBIOS-specific information

The parameters required for SMBIOS with reference values are as follows:

```
#define PLATFORM_OVERRIDE_SMBIOS_SLOT_COUNT 0x1
#define PLATFORM_OVERRIDE_SMBIOS_SLOT0_FAMILY 0x102
#define PLATFORM_OVERRIDE_SMBIOS_SLOT0_CORE_COUNT 16
```

Header file representation:

```
typedef struct {
    uint16_t processor_family;
    uint16_t core_count;
} PE_SMBIOS_TYPE4_INFO;
```

```
typedef struct {
    uint32_t slot_count;
    PE SMBIOS_TYPE4_INFO type4_info[];
} PE_SMBIOS_PROCESSOR_INFO_TABLE;
```

2.1.1.1 Cache configuration

This section provides information on the cache info for each PE:

```
#define PLATFORM_OVERRIDE_CACHE_CNT          0x30
#define PLATFORM_CACHEx_FLAGS                0xFF
#define PLATFORM_CACHEx_OFFSET               0x68
#define PLATFORM_CACHEx_NEXT_LEVEL_INDEX    1
#define PLATFORM_CACHEx_SIZE                 0x10000
#define PLATFORM_CACHEx_CACHE_ID             0x1
#define PLATFORM_CACHEx_IS_PRIVATE           0x1
#define PLATFORM_CACHEx_TYPE                 0
```

2.1.2 PCIe

This section provides information on the number of Peripheral Component Interconnect express (PCIe) root ports and the information required for PCIe enumeration.

```
#define PLATFORM_OVERRIDE_PCIE_ECAM0_HB_COUNT 1
#define PLATFORM_OVERRIDE_PCIE_ECAM0_SEG_NUM 0x0
#define PLATFORM_OVERRIDE_PCIE_ECAM0_START_BUS_NUM 0x0
#define PLATFORM_OVERRIDE_PCIE_ECAM0_END_BUS_NUM 0x8
#define PLATFORM_OVERRIDE_PCIE_ECAM0_EP_BAR64 0x4000100000
#define PLATFORM_OVERRIDE_PCIE_ECAM0_RP_BAR64 0x4000000000
#define PLATFORM_OVERRIDE_PCIE_ECAM0_EP_NPBAR32 0x60000000
#define PLATFORM_OVERRIDE_PCIE_ECAM0_EP_PBAR32 0x60600000
#define PLATFORM_OVERRIDE_PCIE_ECAM0_RP_BAR32 0x60850000
```

PLATFORM_OVERRIDE_PCIE_ECAMx_EP_BAR64:

The address required for 64-bit Prefetchable Memory Base for an PCIe End Point.

PLATFORM_OVERRIDE_PCIE_ECAMx_RP_BAR64:

The address required for 64-bit Prefetchable Memory Base for PCIe Type 1 devices.

PLATFORM_OVERRIDE_PCIE_ECAMx_EP_NPBAR32:

The address required for 32-bit Non-Prefetchable Memory Base for an PCIe End Point.

PLATFORM_OVERRIDE_PCIE_ECAMx_EP_PBAR32:

The address required for 32-bit Prefetchable Memory Base for an PCIe End Point.

PLATFORM_OVERRIDE_PCIE_ECAMx_RP_BAR32:

The address required for 32-bit Memory Base for a PCIe Type 1 devices.

Parameters required for the PCIe enumeration for a platform is populated as follows:

PLATFORM_OVERRIDE_NUM_ECAM:

Represents the number of Enhanced Configuration Access Mechanism (ECAM) regions in the system.

PLATFORM_MAX_HB_COUNT:

Represents the maximum number of Host bridges in the system..

PLATFORM_OVERRIDE_PCIE_ECAM_BASE_ADDR_x:

ECAM base address: ECAM maps PCIe configuration space to a memory address. The memory address to the current configuration space must be provided here.

PLATFORM_OVERRIDE_PCIE_SEGMENT_GRP_NUM_x:

Segment number of the xth ECAM region.

PLATFORM_OVERRIDE_PCIE_START_BUS_NUM_x:

Starting bus number of the xth ECAM region.

PLATFORM_OVERRIDE_PCIE_END_BUS_NUM_x:

Ending bus number of the xth ECAM region.

A platform with one ECAM region is populated as follows:

```
/* PCIe platform config parameters */
#define PLATFORM_OVERRIDE_NUM_ECAM 1

/* Platform config parameters for ECAM_0 */
#define PLATFORM_OVERRIDE_PCIE_ECAM_BASE_ADDR_0 0x60000000
#define PLATFORM_OVERRIDE_PCIE_SEGMENT_GRP_NUM_0 0x0
#define PLATFORM_OVERRIDE_PCIE_START_BUS_NUM_0 0x0
#define PLATFORM_OVERRIDE_PCIE_END_BUS_NUM_0 0xFF
```

Header file representation:

```
typedef struct {
    uint64_t ecam_base; ///< ECAM Base address
    uint32_t segment_num; ///< Segment number of this ECAM
    uint32_t start_bus_num; ///< Start Bus number for this ecam space
    uint32_t end_bus_num; ///< Last Bus number
} PCIE_INFO_BLOCK;

typedef struct {
    uint32_t num_entries;
    PCIE_INFO_BLOCK block[];
} PCIE_INFO_TABLE;
```

2.1.2.1 PCIe device hierarchy table

This hierarchy table is used to obtain platform-specific support such as DMA, P2P and so on.

Parameters to be populated for each PCIe device is as follows:

```
PLATFORM_PCIE_DEVx_CLASSCODE 0x6040000
PLATFORM_PCIE_DEVx_VENDOR_ID 0x13B5
PLATFORM_PCIE_DEVx_DEV_ID 0xDEF
PLATFORM_PCIE_DEVx_BUS_NUM 0
```

```

PLATFORM_PCIE_DEVx_DEV_NUM      1
PLATFORM_PCIE_DEVx_FUNC_NUM     0
PLATFORM_PCIE_DEVx_SEG_NUM      0
PLATFORM_PCIE_DEVx_DMA_SUPPORT  0
PLATFORM_PCIE_DEVx_DMA_COHERENT 0
PLATFORM_PCIE_DEVx_P2P_SUPPORT  1
PLATFORM_PCIE_DEVx_DMA_64BIT    0
PLATFORM_PCIE_DEVx_BEHIND_SMMU  1
PLATFORM_PCIE_DEVx_ATC_SUPPORT  0

```

Header file representation:

```

typedef struct {
    uint64_t class_code;
    uint32_t device_id;
    uint32_t vendor_id;
    uint32_t bus;
    uint32_t dev;
    uint32_t func;
    uint32_t seg;
    uint32_t dma_support;
    uint32_t dma_coherent;
    uint32_t p2p_support;
    uint32_t dma_64bit;
    uint32_t behind_smmu;
    uint32_t atc_present;
    PERIPHERAL_IRQ_MAP irq_map;
} PCIE_READ_BLOCK;

```

2.1.3 DMA

This section provides the configuration options for Direct Memory Access (DMA) controller-based tests. Additionally, it describes the parameters for the number of DMA bus Requesters, and DMA Requester attributes that can be customized.

2.1.3.1 Number of DMA controllers

Header file representation:

```
#define PLATFORM_OVERRIDE_DMA_CNT 0
```

PLATFORM_OVERRIDE_DMA_CNT:

Represents the number of DMA controllers in the system.

2.1.3.2 DMA Requester attributes

Header file representation:

```

typedef struct {
    DMA_INFO_TYPE_e type;
    void            *target;
    void            *port;
    void            *host;
}

```

```
uint32_t      flags;
}DMA_INFO_BLOCK;
```

The actual information stored in the above pointers are implementation-specific.

2.1.4 SMMU and device tests

This section provides an overview on SMMU and the device tests. It also provides information on the number of IOVIRT nodes, SMMUs, RC, Named component, PMCG, ITS blocks, I/O virtualization node-specific information, SMMU node-specific information, RC-specific information, and I/O virtual address mapping.

2.1.4.1 Number of IOVIRT Nodes

Parameters to be filled are:

```
#define IORT_NODE_COUNT 0x13
```

IORT_NODE_COUNT:

Represents the total number of Root Complex (RC), SMMU, ITS, PMCG, and other nodes represented in IORT structure.

2.1.4.2 Number of SMMUs

Parameters to be filled are:

```
#define IOVIRT_SMMUV3_COUNT 5
```

```
#define IOVIRT_SMMUV2_COUNT 0
```

SMMU_COUNT:

Represents the number of SMMUs in the system.

2.1.4.3 Number of RCs

Parameters to be filled are:

```
#define RC_COUNT 0x1
```

RC_COUNT:

Represents the number of RCs present in the system.

2.1.4.4 Number of PMCGs

Parameters to be filled are:

```
#define PMCG_COUNT 0x1
```

PMCG_COUNT:

Represents the number of Performance Monitor Counter Groups (PMCGs) present in the system.

2.1.4.5 Number of named components

Parameters to be filled are:

```
#define IOVIRT_NAMED_COMPONENT_COUNT 2
```

IOVIRT_NAMED_COMPONENT_COUNT

Represents the number of named components present in the system.

2.1.4.6 Number of ITS blocks

Parameters to be filled are:

```
#define IOVIRT_ITS_COUNT 0x1
```

IOVIRT_ITS_COUNT:

Represents the number of Interrupt Translation Service (ITS) nodes in the system.

2.1.4.7 I/O virtualization node-specific information

Header file representation:

```
typedef struct {
    uint32_t type;
    uint32_t num_data_map;
    NODE_DATA data;
    uint32_t flags;
    NODE_DATA_MAP data_map[];
} IOVIRT_BLOCK;

typedef union {
    char name[MAX_NAMED_COMP_LENGTH];
    IOVIRT_RC_INFO_BLOCK rc;
    IOVIRT_PMCG_INFO_BLOCK pmcg;
    uint32_t its_count;
    SMMU_INFO_BLOCK smmu;
} NODE_DATA;
```

2.1.4.8 SMMU node-specific information

Header file representation:

```
typedef struct {
    uint32_t arch_major_rev;    ///< Version 1 or 2 or 3
    uint64_t base;             ///< SMMU controller base address
} SMMU_INFO_BLOCK;
```

IOVIRT_SMMUV3_BASE_ADDRESS:

Represents the SMMU base address in the system.

2.1.4.9 Root Complex node specific information

Header file representation:

```
typedef struct {
    uint32_t segment;
    uint32_t ats_attr;
    uint32_t cca;              ///< Cache Coherency Attribute
    uint64_t smmu_base;
} IOVIRT_RC_INFO_BLOCK;
```

2.1.4.10 PMCG node-specific information

Header file representation:

```
typedef struct {
    uint64_t base;
    uint32_t overflow_gsiv;
    uint32_t node_ref;
} IOVIRT_PMCG_INFO_BLOCK;
```

2.1.4.11 Named component node specific information

Header file representation:

```
typedef struct {
    uint64_t smmu_base; /* SMMU base to which component is attached, else NULL */
    uint32_t cca; /* Cache Coherency Attribute */
    char name[MAX_NAMED_COMP_LENGTH]; /* Device object name */
} IOVIRT_NAMED_COMP_INFO_BLOCK;
```

Named component-specific information on Coresight components

Header file representation

```
typedef struct {
    char identifier[MAX_CS_COMP_LENGTH]; // Hardware ID for Coresight ARM
    implementations
```

```
char dev_name[MAX_CS_COMP_LENGTH]; // Device name of Coresight components
} PLATFORM_OVERRIDE_CORESIGHT_COMP_INFO_BLOCK;

typedef struct {
    PLATFORM_OVERRIDE_CORESIGHT_COMP_INFO_BLOCK component[CS_COMPONENT_COUNT];
} PLATFORM_OVERRIDE_CS_COMP_NODE_DATA;
```

2.1.4.12 I/O virtual address mapping

Header file representation:

```
typedef struct {
    uint32_t input_base;
    uint32_t id_count;
    uint32_t output_base;
    uint32_t output_ref;
} ID_MAP;
```

2.1.5 GIC

This section provides the parameters for Generic Interrupt Controller (GIC) specific test.

GIC-specific tests

Parameters to be filled are:

```
#define PLATFORM_OVERRIDE_GICD_COUNT          0x1
#define PLATFORM_OVERRIDE_GICRD_COUNT         0x1
#define PLATFORM_OVERRIDE_GICITS_COUNT        0x1
#define PLATFORM_OVERRIDE_GICH_COUNT          0x1
#define PLATFORM_OVERRIDE_GICMSIFRAME_COUNT   0x0
#define PLATFORM_OVERRIDE_GICC_TYPE           0x1000
#define PLATFORM_OVERRIDE_GICD_TYPE           0x1001
#define PLATFORM_OVERRIDE_GICC_GICRD_TYPE     0x1002
#define PLATFORM_OVERRIDE_GICR_GICRD_TYPE     0x1003
#define PLATFORM_OVERRIDE_GICITS_TYPE         0x1004
#define PLATFORM_OVERRIDE_GICMSIFRAME_TYPE    0x1005
#define PLATFORM_OVERRIDE_GICH_TYPE           0x1006
#define PLATFORM_OVERRIDE_GICC_BASE            0x30000000
#define PLATFORM_OVERRIDE_GICD_BASE            0x30000000
#define PLATFORM_OVERRIDE_GICRD_BASE          0x300C0000
#define PLATFORM_OVERRIDE_GICITS_BASE         0x30040000
#define PLATFORM_OVERRIDE_GICH_BASE            0x2C010000
#define PLATFORM_OVERRIDE_GICITS_ID           0
#define PLATFORM_OVERRIDE_GICIRD_LENGTH       (0x20000*8)
```

Header file representation:

```
typedef struct {
    uint32_t gic_version;
    uint32_t num_gicc;
    uint32_t num_gicd;
    uint32_t num_gicrd;
    uint32_t num_gicits;
    uint32_t num_gich;
    uint32_t num_msiframes;
    uint32_t gicc_type;
    uint32_t gicd_type;
    uint32_t gicrd_type;
```

```

uint32_t gicrd_length;
uint32_t gicits_type;
uint64_t gicc_base[PLATFORM_OVERRIDE_GICC_COUNT];
uint64_t gicd_base[PLATFORM_OVERRIDE_GICD_COUNT];
uint64_t gicrd_base[PLATFORM_OVERRIDE_GICRD_COUNT];
uint64_t gicits_base[PLATFORM_OVERRIDE_GICITS_COUNT];
uint64_t gicits_id[PLATFORM_OVERRIDE_GICITS_COUNT];
uint64_t gich_base[PLATFORM_OVERRIDE_GICH_COUNT];
uint64_t gicmsiframe_base[PLATFORM_OVERRIDE_GICMSIFRAME_COUNT];
uint64_t gicmsiframe_id[PLATFORM_OVERRIDE_GICMSIFRAME_COUNT];
uint32_t gicmsiframe_flags[PLATFORM_OVERRIDE_GICMSIFRAME_COUNT];
uint32_t gicmsiframe_spi_count[PLATFORM_OVERRIDE_GICMSIFRAME_COUNT];
uint32_t gicmsiframe_spi_base[PLATFORM_OVERRIDE_GICMSIFRAME_COUNT];
} PLATFORM_OVERRIDE_GIC_INFO_TABLE;

```

2.1.6 Timer

This section provides the parameters for timer-specific tests.

2.1.6.1 Timer information

Parameters to be filled are:

```

#define PLATFORM_OVERRIDE_PLATFORM_TIMER_COUNT 0x2
#define PLATFORM_OVERRIDE_S_EL1_TIMER_GSIV 0x1D
#define PLATFORM_OVERRIDE_NS_EL1_TIMER_GSIV 0x1E
#define PLATFORM_OVERRIDE_NS_EL2_TIMER_GSIV 0x1A
#define PLATFORM_OVERRIDE_VIRTUAL_TIMER_GSIV 0x1B
#define PLATFORM_OVERRIDE_EL2_VIR_TIMER_GSIV 28

```

Header file representation:

```

typedef struct {
    uint32_t s_el1_timer_flag;
    uint32_t ns_el1_timer_flag;
    uint32_t el2_timer_flag;
    uint32_t el2_virt_timer_flag;
    uint32_t s_el1_timer_gsiv;
    uint32_t ns_el1_timer_gsiv;
    uint32_t el2_timer_gsiv;
    uint32_t virtual_timer_flag;
    uint32_t virtual_timer_gsiv;
    uint32_t el2_virt_timer_gsiv;
    uint32_t num_platform_timer;
    uint32_t num_watchdog;
    uint32_t sys_timer_status;
} TIMER_INFO_HDR;

typedef struct {
    uint32_t type;
    uint32_t timer_count;
    uint64_t block_cntl_base;
    uint8_t frame_num[8];
    uint64_t GtCntBase[8];
    uint64_t GtCntEl0Base[8];
    uint32_t gsiv[8];
    uint32_t virt_gsiv[8];
    uint32_t flags[8];
} TIMER_INFO_GTBLOCK;

```

```
typedef struct {
    TIMER_INFO_HDR header;
    TIMER_INFO_GTBLOCK gt_info[];
}TIMER_INFO_TABLE;
```

2.1.7 Watchdog timer

This section provides the parameters for the number of watchdog timer tests and watchdog information.

Parameters to be filled are:

```
#define PLATFORM_OVERRIDE_WD_TIMER_COUNT 2
```

2.1.7.1 Watchdog information

The following is the list of watchdog timers present in the system:

- Watchdog timer number
- Control base
- Refresh base
- Interrupt number
- Flags

Header file representation:

```
typedef struct {
    uint64_t wd_ctrl_base;        ///< Watchdog Control Register Frame
    uint64_t wd_refresh_base;     ///< Watchdog Refresh Register Frame
    uint32_t wd_gsiv;             ///< Watchdog Interrupt ID
    uint32_t wd_flags;
}WD_INFO_BLOCK;
```

2.1.8 Memory

This section provides information on the memory map in the system.

PLATFORM_OVERRIDE_MEMORY_ENTRY_COUNT:

Represents the number of memory range entries.

PLATFORM_OVERRIDE_MEMORY_ENTRYx_PHY_ADDR:

Represents the physical address of the xth memory entry.

PLATFORM_OVERRIDE_MEMORY_ENTRYx_VIRT_ADDR:

Represents the virtual address of the xth memory entry.

PLATFORM_OVERRIDE_MEMORY_ENTRYx_SIZE:

Represents the size of the xth memory entry.

PLATFORM_OVERRIDE_MEMORY_ENTRYx_TYPE:

Represents the type of the xth memory entry.

The following is an example for memory map.

```
#define PLATFORM_OVERRIDE_MEMORY_ENTRY_COUNT      0x4
#define PLATFORM_OVERRIDE_MEMORY_ENTRY0_PHY_ADDR  0xC000000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY0_VIRT_ADDR 0xC000000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY0_SIZE      0x4000000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY0_TYPE      MEMORY_TYPE_DEVICE
#define PLATFORM_OVERRIDE_MEMORY_ENTRY1_PHY_ADDR  0x10000000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY1_VIRT_ADDR 0x10000000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY1_SIZE      0xC170000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY1_TYPE      MEMORY_TYPE_NOT_POPULATED
#define PLATFORM_OVERRIDE_MEMORY_ENTRY2_PHY_ADDR  0xFF600000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY2_VIRT_ADDR 0xFF600000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY2_SIZE      0x10000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY2_TYPE      MEMORY_TYPE_RESERVED
#define PLATFORM_OVERRIDE_MEMORY_ENTRY3_PHY_ADDR  0x80000000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY3_VIRT_ADDR 0x80000000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY3_SIZE      0x7F000000
#define PLATFORM_OVERRIDE_MEMORY_ENTRY3_TYPE      MEMORY_TYPE_NORMAL
```

Header file representation:

```
typedef struct {
    MEM_INFO_TYPE_e type;
    uint64_t phy_addr;
    uint64_t virt_addr;
    uint64_t size;
    uint64_t flags; //To Indicate Cacheability etc..
}MEM_INFO_BLOCK;
```

2.1.9 MPAM

This section provides information on the Memory System Resource Partitioning and Monitoring of the system.

Parameters to be filled are:

```
#define MPAM_MAX_MSC_NODE      0x1
#define MPAM_MAX_RSRC_NODE    0x1
#define PLATFORM_MPAM_MSC_COUNT 0x1
#define PLATFORM_MPAM_MSCx_BASE_ADDR 0x1010028000
#define PLATFORM_MPAM_MSCx_ADDR_LEN 0x2004
#define PLATFORM_MPAM_MSCx_MAX_NRDY 10000000
#define PLATFORM_MPAM_MSCx_RSRC_COUNT 0x1
#define PLATFORM_MPAM_MSCx_RSRCx_RIS_INDEX 0x0
#define PLATFORM_MPAM_MSCx_RSRCx_LOCATOR_TYPE 0x1
#define PLATFORM_MPAM_MSCx_RSRCx_DESCRIPTOR1 0x0
#define PLATFORM_MPAM_MSCx_RSRCx_DESCRIPTOR2 0x0
```

Header file representation:

```

*
* @brief Mpam Resource Node
*/

typedef struct {
    uint8_t ris_index;
    uint8_t locator_type;
    uint64_t descriptor1;
    uint32_t descriptor2;
} MPAM_RESOURCE_NODE;
/*
* @brief Mpam MSC Node
*/

typedef struct {
    uint8_t   intf_type;
    uint32_t  identifier;
    uint64_t  msc_base_addr;

    uint32_t  msc_addr_len;
    uint32_t  max_nrdy;

    uint32_t  rsrc_count;
    MPAM_RESOURCE_NODE rsrc_node[];
} MPAM_MSC_NODE;
/*
* @brief Mpam info table
*/

typedef struct {
    uint32_t msc_count;
    MPAM_MSC_NODE msc_node[];
} MPAM_INFO_TABLE;

```

2.1.9.1 SRAT

This section provides information on the System Affinity table of the system.

Parameters to be filled are:

```

#define PLATFORM_OVERRIDE_NUM_SRAT_ENTRIES 17
#define PLATFORM_OVERRIDE_MEM_AFF_CNT      1
#define PLATFORM_OVERRIDE_GICC_AFF_CNT     16

```

The memory affinity and GICC Affinity parameters to be filled are:

```

#define PLATFORM_SRAT_MEMX_PROX_DOMAIN 0x0
#define PLATFORM_SRAT_MEMX_FLAGS      0x1
#define PLATFORM_SRAT_MEMX_ADDR_BASE  0x8080000000
#define PLATFORM_SRAT_MEMX_ADDR_LEN   0x3F7F7FFFFFFF
#define PLATFORM_SRAT_GICCX_PROX_DOMAIN 0x0
#define PLATFORM_SRAT_GICCX_PROC_UID   0x0
#define PLATFORM_SRAT_GICCX_FLAGS      0x1
#define PLATFORM_SRAT_GICCX_CLK_DOMAIN 0x0

```

Header file representation :

```
typedef union {
    SRAT_MEM_AFF_ENTRY mem_aff;
    SRAT_GICC_AFF_ENTRY gicc_aff;
} SRAT_NODE_INFO;

typedef struct {
    uint32_t node_type; /* Node type*/
    SRAT_NODE_INFO node_data;
} SRAT_INFO_ENTRY;

typedef struct {
    uint32_t num_of_srat_entries;
    uint32_t num_of_mem_ranges;
    SRAT_INFO_ENTRY srat_info[];
} SRAT_INFO_TABLE;
```

2.1.9.2 PCC

Parameters to be filled are:

```
#define PLATFORM_PCC_SUBSPACE_COUNT 0x1
#define PLATFORM_PCC_SUBSPACE0_INDEX 0x0
#define PLATFORM_PCC_SUBSPACE0_TYPE 0x3
#define PLATFORM_PCC_SUBSPACE0_BASE 0x0
#define PLATFORM_PCC_SUBSPACE0_DOORBELL_PRESERVE 0x0
#define PLATFORM_PCC_SUBSPACE0_DOORBELL_WRITE 0x0
#define PLATFORM_PCC_SUBSPACE0_MIN_REQ_TURN_TIME 0x0
#define PLATFORM_PCC_SUBSPACE0_CMD_COMPLETE_CHK_MASK 0x0
#define PLATFORM_PCC_SUBSPACE0_CMD_UPDATE_PRESERVE 0x0
#define PLATFORM_PCC_SUBSPACE0_CMD_COMPLETE_UPDATE_SET 0x0
```

Following fields follow `GENERIC_ADDRESS_STRUCTURE` defined in `platform_override_sbsa_struct.h`

```
#define PLATFORM_PCC_SUBSPACE0_DOORBELL_REG {0x0, 0x0, 0x0, 0x0, 0xDEADDEAD}
#define PLATFORM_PCC_SUBSPACE0_CMD_COMPLETE_UPDATE_REG {0x0, 0x0, 0x0, 0x0, 0xDEADDEAD}
#define PLATFORM_PCC_SUBSPACE0_CMD_COMPLETE_CHK_REG {0x0, 0x0, 0x0, 0x0, 0xDEADDEAD}
```

Header file representation :

```
typedef struct {
    uint8_t addr_space_id;
    uint8_t reg_bit_width;
    uint8_t reg_bit_offset;
    uint8_t access_size;
    uint64_t addr;
} GENERIC_ADDRESS_STRUCTURE;

typedef struct {
    uint64_t base_addr; /* base addr of shared mem-region */
    GENERIC_ADDRESS_STRUCTURE doorbell_reg; /* doorbell register */
    uint64_t doorbell_preserve; /* doorbell register preserve mask */
    uint64_t doorbell_write; /* doorbell register set mask */
}
```

```

uint32_t min_req_turnaround_usec;          /* minimum request turnaround
time */
GENERIC_ADDRESS_STRUCTURE cmd_complete_chk_reg; /* command complete check
register */
uint64_t cmd_complete_chk_mask;            /* command complete check mask */
GENERIC_ADDRESS_STRUCTURE cmd_complete_update_reg; /* command complete update
register */
uint64_t cmd_complete_update_preserve;      /* command complete update
preserve */
uint64_t cmd_complete_update_set;          /* command complete update set
mask */
} PLATFORM_OVERRIDE_PCC_SUBSPACE_TYPE_3;

typedef union {
PLATFORM_OVERRIDE_PCC_SUBSPACE_TYPE_3 pcc_ss_type_3; /* PCC type 3 info */
} PLATFORM_OVERRIDE_PCC_TYPE_SPECIFIC_INFO;

typedef struct {
uint32_t subspace_idx;                      /* PCC subspace index in PCCT
ACPI table */
uint32_t subspace_type;                    /* type of PCC subspace */

PLATFORM_OVERRIDE_PCC_TYPE_SPECIFIC_INFO
type_spec_info;                            /* PCC subspace type specific
info */
} PLATFORM_OVERRIDE_PCC_INFO;

typedef struct {
uint32_t subspace_cnt;                      /* number of PCC subspace info
stored */
PLATFORM_OVERRIDE_PCC_INFO pcc_info[PLATFORM_PCC_SUBSPACE_COUNT];
/* array of PCC info blocks */
} PLATFORM_OVERRIDE_PCC_INFO_TABLE;

```

2.1.10 HMAT

This sections provides information on the Heterogeneous Memory Attribute Table

Parameters required to be populated are:

```

#define PLATFORM_OVERRIDE_HMAT_MEM_ENTRIES    0x4
#define HMAT_NODE_MEM_SLLBIC                 0x1
#define HMAT_NODE_MEM_SLLBIC_DATA_TYPE      0x3
#define HMAT_NODE_MEM_SLLBIC_FLAGS          0x0
#define HMAT_NODE_MEM_SLLBIC_ENTRY_BASE_UNIT 0x64
#define PLATFORM_HMAT_MEMx_PROX_DOMAIN      0x0
#define PLATFORM_HMAT_MEMx_MAX_WRITE_BW     0x82
#define PLATFORM_HMAT_MEMx_MAX_READ_BW      0x82

```

Header file representation:

```

typedef struct {
uint32_t mem_prox_domain;                  /* Proximity domain of the memory region*/
uint64_t write_bw;                         /* Maximum write bandwidth */
uint64_t read_bw;                         /* Maximum read bandwidth */
} HMAT_BW_ENTRY;

typedef struct {
uint32_t num_of_mem_prox_domain;           /* Number of Memory Proximity Domains */
HMAT_BW_ENTRY bw_info[];                  /* Array of bandwidth info based on
proximity domain */

```

```
} HMAT_INFO_TABLE;
```

2.1.11 RAS

This section provides Information on the Reliability, Availability and Serviceability features of the system.

Parameters to be filled are:

```
#define PLATFORM_OVERRIDE_NUM_RAS_NODES      0x1
#define PLATFORM_OVERRIDE_NUM_PE_RAS_NODES   0x1
#define PLATFORM_OVERRIDE_NUM_MC_RAS_NODES    0x0
#define RAS2_MAX_NUM_BLOCKS                  0x4
```

Header file representation:

```
typedef struct {
    RAS_NODE_TYPE_e type;           /* Node Type PE/GIC/SMMU */
    uint16_t length;               /* Length of the Node */
    uint64_t num_intr_entries;      /* Number of Interrupt Entry */
    RAS_NODE_DATA node_data;        /* Node Specific Data */
    RAS_INTERFACE_INFO intf_info;   /* Node Interface Info */
    RAS_INTERRUPT_INFO intr_info[2]; /* Node Interrupt Info */
} RAS_NODE_INFO;

typedef struct {
    uint32_t num_nodes;             /* Number of total RAS Nodes */
    uint32_t num_pe_node;           /* Number of PE RAS Nodes */
    uint32_t num_mc_node;           /* Number of Memory Controller Nodes */
    RAS_NODE_INFO node[];           /* Array of RAS nodes */
} RAS_INFO_TABLE;

typedef struct {
    RAS2_FEAT_TYPE type;            /* RAS2 feature type */
    RAS2_BLOCK_INFO block_info;     /* RAS2 block info */
} RAS2_BLOCK;

typedef struct {
    uint32_t num_all_block;         /* Number of RAS2 feature blocks */
    uint32_t num_of_mem_block;      /* Number of memory feature blocks */
    RAS2_BLOCK blocks[];
} RAS2_INFO_TABLE;
```

2.1.12 PMU

This section provides Information on the Performance Monitoring Unit of the system.

Parameters to be filled are:

```
#define MAX_NUM_OF_PMU_SUPPORTED      512
#define PLATFORM_OVERRIDE_PMU_NODE_CNT 0x1
#define PLATFORM_PMU_NODEx_BASE0     0x1010028000
#define PLATFORM_PMU_NODEx_BASE1     0x0
```

```
#define PLATFORM_PMU_NODEx_TYPE          0x2
#define PLATFORM_PMU_NODEx_PRI_INSTANCE  0x0
#define PLATFORM_PMU_NODEx_SEC_INSTANCE  0x0
#define PLATFORM_PMU_NODEx_DUAL_PAGE_EXT  0x0
```

Header file representation:

```
typedef struct {
    uint8_t type; /* The component that this PMU block is
                  associated with*/
    uint64_t primary_instance; /* Primary node instance, specific to the PMU
                              type*/
    uint32_t secondary_instance; /* Secondary node instance, specific to the PMU
                                type*/
    uint8_t dual_page_extension; /* Support of the dual-page mode*/
    uint64_t base0; /* Base address of Page 0 of the PMU*/
    uint64_t base1; /* Base address of Page 1 of the PMU,
                    valid only if dual_page_extension is 1*/
} PMU_INFO_BLOCK;

typedef struct {
    uint32_t pmu_count; /* Total number of PMU info blocks*/
    PMU_INFO_BLOCK info[]; /* PMU info blocks for each PMU nodes*/
} PMU_INFO_TABLE;
```

2.1.13 MMU Configuration

This section provides information on the MMU for the PE MMU.

The parameters required for the PE MMU are populated as follows:

```
#define PLATFORM_PAGE_SIZE      0x1000
#define PLATFORM_OVERRIDE_MMU_PGT_IAS  48
#define PLATFORM_OVERRIDE_MMU_PGT_OAS  48
```

2.1.14 Peripherals

This section provides information on the peripherals in the system.

Parameters to be filled are:

```
#define PLATFORM_OVERRIDE_PERIPHERAL_COUNT 3 //UART + USB + SATA
#define UART_ADDRESS 0xF98DFE18
#define BASE_ADDRESS_ADDRESS_SPACE_ID      0x0
#define BASE_ADDRESS_REGISTER_BIT_WIDTH    0x20
#define BASE_ADDRESS_REGISTER_BIT_OFFSET   0x0
#define BASE_ADDRESS_ADDRESS_SIZE          0x3
#define BASE_ADDRESS_ADDRESS              0x2A400000
#define UART_BAUD_RATE                     115200
#define UART_CLK_RATE_HZ                   48000000
#define UART_INTERRUPT_TYPE                8
#define UART_IRQ                           0
#define UART_GLOBAL_SYSTEM_INTERRUPT        0x70
#define UART_PCI_DEVICE_ID                 0xFFFF
#define UART_PCI_VENDOR_ID                 0xFFFF
#define UART_PCI_BUS_NUMBER                 0x0
#define UART_PCI_DEV_NUMBER                0x0
```

```
#define UART_PCI_FUNC_NUMBER      0x0
#define UART_PCI_FLAGS            0x0
#define UART_PCI_SEGMENT          0x0
```

**Note**

Ensure that the `BASE_ADDRESS_ADDRESS`, `UART_BAUD_RATE`, and `UART_CLK_RATE_HZ` are configured properly to get PL011 UART prints on the console.

2.2 Bare-metal Boot

This section provides information on the Bare-metal boot requirements of the system.

The following system-specific definitions must be filled to load bootable image.

Parameters to be filled are:

```
#define PLATFORM_OVERRIDE_WORLD_IMAGE_SIZE  0x4000000
#define PLARFORM_MEMORY_POOL_SIZE           (250 * 100000)
#define PLATFORM_SHARED_MEMORY_REGION       0x100000
#define PLATFORM_NORMAL_WORLD_IMAGE_BASE    0xE0000000
```

For more information on how to run SBSA ACS with bootwrapper code, see the [README](#).

**Note**

`PLATFORM_NORMAL_WORLD_IMAGE_BASE` is the entry point to BL33.

3. Porting requirements

This chapter provides information on different PAL APIs in PE, GIC, timer, IOVIRT, PCIe, SMMU, peripheral, DMA, PMU, MPAM, RAS, exerciser, and other miscellaneous APIs.

3.1 PAL implementation

PAL is a C-based, Arm-defined API that you can implement. Each test platform requires a PAL implementation of its own.

The bare-metal reference code provides a reference implementation for a subset of APIs. Additional code must be implemented to match the target SoC implementation under the tests.



Note

There are two implementation types for the PAL APIs and are classified in the following tables:

- Yes: indicates that the implementation of this API is already present. Since the values are platform-specific, it must be taken from the platform configuration file.
- Platform-specific: you must implement all the APIs that are marked as platform-specific.

3.1.1 PE

The following table lists the different types of APIs in PE.

Table 3-1: PE APIs and their details

API name	Function prototype	Implementation
create_info_table	<code>void pal_pe_create_info_table(PE_INFO_TABLE *PeTable);</code>	Yes
call_smc	<code>void pal_pe_call_smc(ARM_SMC_ARGS *args);</code>	Yes
execute_payload	<code>void pal_pe_execute_payload(ARM_SMC_ARGS *args);</code>	Yes
update_elr	<code>void pal_pe_update_elr(void *context, uint64_t offset);</code>	Platform-specific
get_esr	<code>uint64_t pal_pe_get_esr(void *context);</code>	Platform-specific
data_cache_ops_by_va	<code>void pal_pe_data_cache_ops_by_va(uint64_t addr, uint32_t type);</code>	Yes
get_far	<code>uint64_t pal_pe_get_far(void *context);</code>	Platform-specific
install_esr	<code>uint32_t pal_pe_install_esr(uint32_t exception_type, void(*esr)(uint64_t, void *));</code>	Platform-specific
get_num	<code>uint32_t pal_pe_get_num();</code>	Yes

API name	Function prototype	Implementation
psci_get_conduit	<code>uint32_t pal_psci_get_conduit(void)</code>	Platform-specific

3.1.2 GIC

The following table lists the different types of APIs in GIC.

Table 3-2: GIC APIs and their details

API name	Function prototype	Implementation
create_info_table	<code>void pal_gic_create_info_table(GIC_INFO_TABLE* gic_info_table);</code>	Yes
install_isr	<code>uint32_t pal_gic_install_isr(uint32_t int_id, void(*isr)(void));</code>	Platform-specific
end_of_interrupt	<code>uint32_t pal_gic_end_of_interrupt(uint32_t int_id);</code>	Platform-specific
request_irq	<code>uint32_t pal_gic_request_irq(unsigned int irq_num, unsigned int mapped_irq_num, void *isr);</code>	Platform-specific
free_irq	<code>void pal_gic_free_irq(unsigned int irq_num, unsigned int mapped_irq_num);</code>	Platform-specific
set_intr_trigger	<code>uint32_t pal_gic_set_intr_trigger(uint32_t int_id INTR_TRIGGER_INFO_TYPE_e trigger_type);</code>	Platform-specific
get_num_nongic_ctrl	<code>uint32_t pal_get_num_nongic_ctrl(void);</code>	Platform-specific

3.1.3 Timer

The following table lists the different types of APIs in timer.

Table 3-3: Timer APIs and their details

API name	Function prototype	Implementation
create_info_table	<code>void pal_timer_create_info_table(TIMER_INFO_TABLE *timer_info_table);</code>	Yes
wd_create_info_table	<code>void pal_wd_create_info_table(WD_INFO_TABLE *wd_table);</code>	Yes
get_counter_frequency	<code>uint64_t pal_timer_get_counter_frequency(void);</code>	Yes

3.1.4 IOVIRT

The following table lists the different types of APIs in IOVIRT.

Table 3-4: IOVIRT APIs and their details

API name	Function prototype	Implementation
create_info_table	<code>void pal_iovirt_create_info_table(IOVIRT_INFO_TABLE *iovirt);</code>	Yes
unique_rid_strid_map	<code>uint32_t pal_iovirt_unique_rid_strid_map(uint64_t rc_block);</code>	Yes

API name	Function prototype	Implementation
check_unique_ctx_initd	uint32_t pal_iovirt_check_unique_ctx_intid(uint64_t smmu_block);	Yes
get_rc_smmu_base	uint64_t pal_iovirt_get_rc_smmu_base(IOVIRT_INFO_TABLE *iovirt, uint32_t rc_seg_num, uint32_t rid);	Yes

3.1.5 PCIe

The following table lists the different types APIs in PCIe.

Table 3-5: PCIe APIs and their details

API name	Function prototype	Implementation
create_info_table	void pal_pcie_create_info_table (PCIE_INFO_TABLE *PcieTable);	Yes
read_cfg	uint32_t pal_pcie_read_cfg(uint32_t bdf, uint32_t offset, uint32_t *data);	Yes
get_msi_vectors	uint32_t pal_get_msi_vectors(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn, PERIPHERAL_VECTOR_LIST**mvector);	Platform-specific
get_pcie_type	uint32_t pal_pcie_get_pcie_type(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);	Yes
p2p_support	uint32_t pal_pcie_p2p_support(void);	Yes
read_ext_cap_word	void pal_pcie_read_ext_cap_word(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn, uint32_t ext_cap_id, uint8_t offset, uint16_t *val);	Yes
get_bdf_wrapper	uint32_t pal_pcie_get_bdf_wrapper (uint32_t class_code, uint32_t start_bdf);	Yes
bdf_to_dev	void *pal_pci_bdf_to_dev(uint32_t bdf);	Yes
pal_pcie_ecam_base	uint64_t pal_pcie_ecam_base(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t func);	Yes
pci_cfg_read	uint32_t pal_pci_cfg_read(uint32_t bus, uint32_t dev, uint32_t func, uint32_t offset, uint32_t *value)	Yes
pci_cfg_write	void pal_pci_cfg_write(uint32_t bus, uint32_t dev, uint32_t func, uint32_t offset, uint32_t data)	Yes
program_bar_reg	void pal_pcie_program_bar_reg(uint32_t bus, uint32_t dev, uint32_t func);	Yes
enumerate_device	uint32_t pal_pcie_enumerate_device(uint32_t bus, uint32_t sec_bus)	Yes
get_bdf	uint32_t pal_pcie_get_bdf(uint32_t ClassCode, uint32_t StartBdf)	Yes
increment_bus_dev	uint32_t pal_increment_bus_dev(uint32_t StartBdf)	Yes
get_base	uint64_t pal_pcie_get_base(uint32_t bdf, uint32_t bar_index)	Yes
io_read_cfg	uint32_t pal_pcie_io_read_cfg(uint32_t Bdf, uint32_t offset, uint32_t *data) ;	Yes
io_write_cfg	void pal_pcie_io_write_cfg(uint32_t bdf, uint32_t offset, uint32_t data);	Yes
get_snoop_bit	uint32_t pal_pcie_get_snoop_bit(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);	Yes

API name	Function prototype	Implementation
is_device_behind_smmu	<code>uint32_t pal_pcie_is_device_behind_smmu(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);</code>	Yes
get_dma_support	<code>uint32_t pal_pcie_get_dma_support(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);</code>	Yes
get_dma_coherent	<code>uint32_t pal_pcie_get_dma_coherent(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);</code>	Yes
is_devicedma_64bit	<code>uint32_t pal_pcie_is_devicedma_64bit(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);</code>	Yes
get_legacy_irq_map	<code>uint32_t pal_pcie_get_legacy_irq_map(uint32_t Seg, uint32_t Bus, uint32_t Dev, uint32_t Fn, PERIPHERAL_IRQ_MAP *IrqMap);</code>	Platform-specific
get_root_port_bdf	<code>uint32_t pal_pcie_get_root_port_bdf(uint32_t *Seg, uint32_t *Bus, uint32_t *Dev, uint32_t *Func);</code>	Yes
dev_p2p_support	<code>uint32_t pal_pcie_dev_p2p_support(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);</code>	Yes
is_cache_present	<code>uint32_t pal_pcie_is_cache_present(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);</code>	Yes
is_onchip_peripheral	<code>uint32_t pal_pcie_is_onchip_peripheral(uint32_t bdf);</code>	Platform-specific
check_device_list	<code>uint32_t pal_pcie_check_device_list(void);</code>	Yes
get_rp_transaction_frwd_support	<code>uint32_t pal_pcie_get_rp_transaction_frwd_support(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);</code>	Platform-specific
check_device_valid	<code>uint32_t pal_pcie_check_device_valid(uint32_t bdf);</code>	Platform-specific
mem_get_offset	<code>uint32_t pal_pcie_mem_get_offset(uint32_t bdf, PCIE_MEM_TYPE_INFO emem_type);</code>	Platform-specific
bar_mem_read	<code>uint32_t pal_pcie_bar_mem_read(uint32_t Bdf, uint64_t address, uint32_t *data);</code>	Yes
bar_mem_write	<code>uint32_t pal_pcie_bar_mem_write(uint32_t Bdf, uint64_t address, uint32_t data);</code>	Yes
dsm_ste_tags	<code>uint32_t pal_pcie_dsm_ste_tags(void);</code>	No

3.1.6 SMMU

The following table lists the different types of APIs in SMMU.

Table 3-6: SMMU APIs and their details

API name	Function prototype	Implementation
check_device_iova	<code>uint32_t pal_smmu_check_device_iova(void *port, uint64_t dma_addr);</code>	Platform-specific
device_start_monitor_iova	<code>void pal_smmu_device_start_monitor_iova(void *port);</code>	Platform-specific
device_stop_monitor_iova	<code>void pal_smmu_device_stop_monitor_iova(void *port);</code>	Platform-specific
pa2iova	<code>uint64_t pal_smmu_pa2iova(uint64_t smmu_base, uint64_t pa);</code>	Platform-specific

API name	Function prototype	Implementation
smmu_disable	<code>uint32_t pal_smmu_disable(uint64_t smmu_base);</code>	Platform-specific
create_pasid_entry	<code>uint32_t pal_smmu_create_pasid_entry(uint64_t smmu_base, uint32_t pasid);</code>	Platform-specific
get_device_path	<code>uint32_t pal_get_device_path(const char *hid, char hid_path[][MAX_NAMED_COMP_LENGTH]);</code>	Yes
is_etr_behind_catu	<code>uint32_t pal_smmu_is_etr_behind_catu(char *etr_path);</code>	Platform-specific

3.1.7 Peripheral

The following table lists the different types of APIs in peripheral.

Table 3-7: Peripheral APIs and their details

API name	Function prototype	Implementation
create_info_table	<code>void pal_peripheral_create_info_table(PERIPHERAL_INFO_TABLE *per_info_table);</code>	Yes
is_pcie	<code>uint32_t pal_peripheral_is_pcie(uint32_t seg, uint32_t bus, uint32_t dev, uint32_t fn);</code>	Yes
memory_create_info_table	<code>void pal_memory_create_info_table(MEMORY_INFO_TABLE *memoryInfoTable);</code>	Platform-specific
memory_ioremap	<code>uint64_t pal_memory_ioremap(void *addr, uint32_t size, uint32_t attr);</code>	Platform-specific
memory_unmap	<code>void pal_memory_unmap(void *addr);</code>	Platform-specific
memory_get_unpopulated_addr	<code>uint64_t pal_memory_get_unpopulated_addr(uint64_t *addr, uint32_t instance)</code>	Platform-specific

3.1.8 MPAM

The following table lists the different types of APIs in MPAM:

Table 3-8: MPAM APIs and their details

API name	Functional prototype	Implementation
create_info_table	<code>void pal_mpam_create_info_table(MPAM_INFO_TABLE *MpamTable);</code>	Yes
create_info_table	<code>void pal_hmat_create_info_table(HMAT_INFO_TABLE *HmatTable);</code>	Yes
create_info_table	<code>void pal_srat_create_info_table(SRAT_INFO_TABLE *SratTable);</code>	Yes

API name	Functional prototype	Implementation
create_info_table	<code>void pal_cache_create_info_table(CACHE_INFO_TABLE *CacheTable, PE_INFO_TABLE *PeTable);</code>	Yes
create_info_table	<code>void pal_pcc_create_info_table(PCC_INFO_TABLE *PccInfoTable);</code>	Yes

3.1.9 RAS

The following table lists the different types of APIs in RAS:

Table 3-9: RAS APIs and their details

API name	Function prototype	Implementation
ras_create_info_table	<code>void pal_ras_create_info_table(RAS_INFO_TABLE *RasInfoTable);</code>	Yes
ras2_create_info_table	<code>void pal_ras2_create_info_table(RAS2_INFO_TABLE *ras2_info_table);</code>	Yes
setup_error	<code>uint32_t pal_ras_setup_error(RAS_ERR_IN_t in_param, RAS_ERR_OUT_t *out_param);</code>	Platform-specific
inject_error	<code>uint32_t pal_ras_inject_error(RAS_ERR_IN_t in_param, RAS_ERR_OUT_t *out_param);</code>	Platform-specific
wait_timeout	<code>void pal_ras_wait_timeout(uint32_t count);</code>	Platform-specific
check_plat_poison_support	<code>uint32_t pal_ras_check_plat_poison_support();</code>	Platform-specific

3.1.10 DMA

The following table lists the different types of APIs in DMA.

Table 3-10: DMA APIs and their details

API name	Function prototype	Implementation
create_info_table	<code>void pal_dma_create_info_table(DMA_INFO_TABLE *dma_info_table);</code>	Yes
start_from_device	<code>uint32_t pal_dma_start_from_device(void *dma_target_buf, uint32_t length, void *host, void *dev);</code>	Platform-specific
start_to_device	<code>uint32_t pal_dma_start_to_device(void *dma_source_buf, uint32_t length, void *host, void *target, uint32_t timeout);</code>	Platform-specific
mem_alloc	<code>uint64_t pal_dma_mem_alloc(void *buffer, uint32_t length, void *dev, uint32_t flags);</code>	Platform-specific
scsi_get_dma_addr	<code>void pal_dma_scsi_get_dma_addr(void *port, void *dma_addr, uint32_t *dma_len);</code>	Platform-specific
mem_get_attrs	<code>int pal_dma_mem_get_attrs(void *buf, uint32_t *attr, uint32_t *sh)</code>	Platform-specific
dma_mem_free	<code>void pal_dma_mem_free(void *buffer, addr_t mem_dma, unsigned int length, void *port, unsigned int flags);</code>	Platform-specific

3.1.11 Exerciser

The following table lists the different types of APIs in exerciser.

Table 3-11: Exerciser APIs and their details

API name	Function prototype	Implementation
get_ecsr_base	uint64_t pal_exerciser_get_ecsr_base(uint32_t Bdf, uint32_t BarIndex)	Platform-specific
get_pcie_config_offset	uint64_t pal_exerciser_get_pcie_config_offset(uint32_t Bdf)	Platform-specific
start_dma_direction	uint32_t pal_exerciser_start_dma_direction(uint64_t Base, EXERCISER_DMA_ATTRDirection)	Platform-specific
find_pcie_capability	uint32_t pal_exerciser_find_pcie_capability(uint32_t ID, uint32_t Bdf, uint32_t Value, uint32_t *Offset)	Platform-specific
set_param	uint32_t pal_exerciser_set_param(EXERCISER_PARAM_TYPE type, uint64_t value1, uint64_t value2, uint32_t bdf);	Platform-specific
get_param	uint32_t pal_exerciser_get_param(EXERCISER_PARAM_TYPE type, uint64_t *value1, uint64_t *value2, uint32_t bdf);	Platform-specific
set_state	uint32_t pal_exerciser_set_state(EXERCISER_STATE state, uint64_t *value, uint32_t bdf);	Platform-specific
get_state	uint32_t pal_exerciser_get_state(EXERCISER_STATE *state, uint32_t bdf);	Platform-specific
ops	uint32_t pal_exerciser_ops(EXERCISER_OPS ops, uint64_t param, uint32_t instance);	Platform-specific
get_data	uint32_t pal_exerciser_get_data(EXERCISER_DATA_TYPE type, exerciser_data_t *data, uint32_t bdf, uint64_t ecam);	Platform-specific
is_bdf_exerciser	uint32_t pal_is_bdf_exerciser(uint32_t bdf)	Platform-specific
disable_rp_pio_register	void pal_exerciser_disable_rp_pio_register(uint32_t bdf)	Platform-specific
check_poison_data_forwarding_support	uint32_t pal_exerciser_check_poison_data_forwarding_support()	Platform-specific
get_pcie_ras_compliant_err_node	uint32_t pal_exerciser_get_pcie_ras_compliant_err_node(uint32_t bdf, uint32_t rp_bdf)	Platform-specific
get_ras_status	uint64_t pal_exerciser_get_ras_status(uint32_t ras_node, uint32_t bdf, uint32_t rp_bdf)	Platform-specific
set_bar_response	uint32_t pal_exerciser_set_bar_response(uint32_t bdf)	Platform-specific

3.1.12 Memory map

The following table lists the different types of APIs required for Memory Map.

Table 3-12: Miscellaneous APIs and their descriptions

API name	Function prototype	Description
add_mmap	<code>void pal_mmu_add_mmap(void);</code>	Platform-specific
get_mmap_list	<code>void *pal_mmu_get_mmap_list(void);</code>	Platform-specific
get_mapping_count	<code>uint32_t pal_mmu_get_mapping_count(void);</code>	Platform-specific

3.1.13 Miscellaneous

The following table lists the different types of miscellaneous PAL APIs.

Table 3-13: Miscellaneous APIs and their details

API name	Function prototype	Implementation
mmio_read8	<code>uint8_t pal_mmio_read8(uint64_t addr);</code>	Yes
mmio_read16	<code>uint16_t pal_mmio_read16(uint64_t addr);</code>	Yes
mmio_read	<code>uint32_t pal_mmio_read(uint64_t addr);</code>	Yes
mmio_read64	<code>uint64_t pal_mmio_read64(uint64_t addr);</code>	Yes
mmio_write8	<code>void pal_mmio_write8(uint64_t addr, uint8_t data);</code>	Yes
mmio_write16	<code>void pal_mmio_write16(uint64_t addr, uint16_t data);</code>	Yes
mmio_write	<code>void pal_mmio_write(uint64_t addr, uint32_t data);</code>	Yes
mmio_write64	<code>void pal_mmio_write64(uint64_t addr, uint64_t data);</code>	Yes
print	<code>void pal_print(char8_t *string, uint64_t data);</code>	Platform-specific
uart_print	<code>void pal_uart_print(int log, const char *fmt, ...);</code>	Yes
print_raw	<code>void pal_print_raw(uint64_t addr, char *string, uint64_t data)</code>	Yes
mem_free	<code>void pal_mem_free(void *buffer);</code>	Platform-specific
mem_compare	<code>int pal_mem_compare(void *src, void *dest, uint32_t len);</code>	Yes
mem_set	<code>void pal_mem_set(void *buf, uint32_t size, uint8_t value);</code>	Yes
mem_allocate_shared	<code>void pal_mem_allocate_shared(uint32_t num_pe, uint32_t sizeofentry);</code>	Yes
mem_get_shared_addr	<code>uint64_t pal_mem_get_shared_addr(void);</code>	Yes
mem_free_shared	<code>void pal_mem_free_shared(void);</code>	Yes
mem_alloc	<code>void *pal_mem_alloc(uint32_t size);</code>	Yes
mem_virt_to_phys	<code>void *pal_mem_virt_to_phys(void *va);</code>	Platform-specific
mem_alloc_cacheable	<code>void *pal_mem_alloc_cacheable(uint32_t Bdf, uint32_t Size, void **Pa);</code>	Platform-specific
mem_free_cacheable	<code>void pal_mem_free_cacheable(uint32_t Bdf, uint32_t Size, void *Va, void *Pa);</code>	Platform-specific

API name	Function prototype	Implementation
mem_phys_to_virt	<code>void *pal_mem_phys_to_virt (uint64_t Pa);</code>	Platform-specific
strncmp	<code>uint32_t pal_strncmp(const char8_t *str1, const char8_t *str2, uint32_t len);</code>	Yes
memcpy	<code>void *pal_memcpy(void *DestinationBuffer, const void *SourceBuffer, uint32_t Length);</code>	Yes
time_delay_ms	<code>uint64_t pal_time_delay_ms(uint64_t time_ms);</code>	Platform-specific
page_size	<code>uint32_t pal_mem_page_size();</code>	Platform-specific
alloc_pages	<code>void *pal_mem_alloc_pages (uint32 NumPages);</code>	Platform-specific
free_pages	<code>void pal_mem_free_pages (void *PageBase, uint32_t NumPages);</code>	Platform-specific
mem_calloc	<code>void *pal_mem_calloc(uint32_t num, uint32_t Size);</code>	Yes
aligned_alloc	<code>void *pal_aligned_alloc(uint32_t alignment, uint32_t size);</code>	Yes
mem_free_aligned	<code>void pal_mem_free_aligned(void *buffer);</code>	Platform-specific
strncpy	<code>void *pal_strncpy(void *DestinationStr, const void *SourceStr, uint32_t Length);</code>	Yes
uart_pl011_putc	<code>void pal_driver_uart_pl011_putc(int c);</code>	Yes

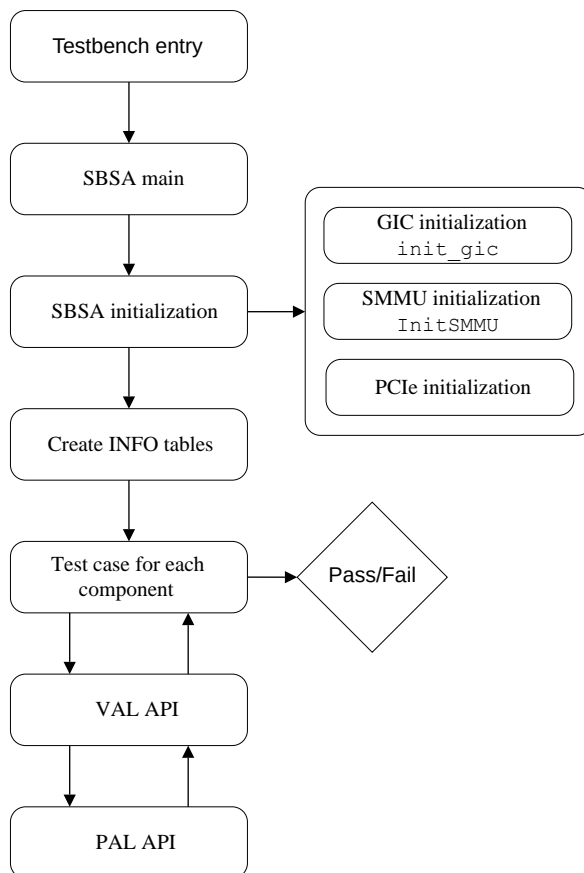
4. SBSA ACS flow

This chapter provides the SBSA ACS flow diagram and SBSA test example flow.

4.1 SBSA ACS flow diagram

The following flow diagram shows the sequence of events from initialization of devices, initialization of SBSA test data structures, and test case execution.

Figure 4-1: SBSA flow diagram

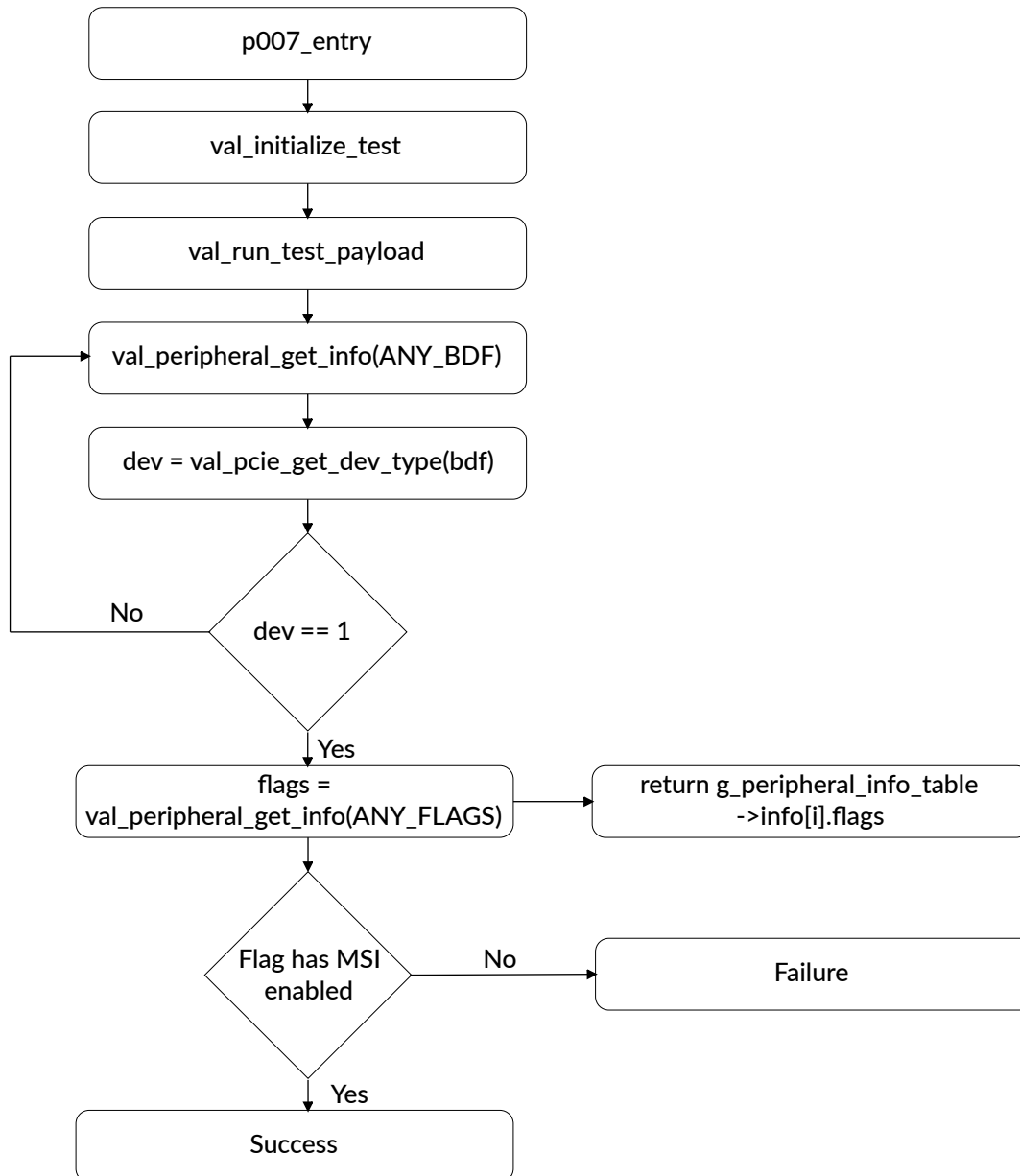


4.2 SBSA test example flow

If the device is Message-Signaled Interrupt (MSI) enabled, then the flag is set to MSI_ENABLED by the PAL layer. The test checks whether the device is of type endpoint and then checks if the flags are set to MSI_ENABLED.

The following flowchart shows the test that checks MSI support in a PCIe device.

Figure 4-2: SBSA example flow diagram



Proprietary Notice

This document is protected by copyright and other related rights and the use 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 Limited ("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 the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. 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, any patents, copyrights, trade secrets, trademarks, or other rights.

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.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted 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.

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 this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

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

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in the Arm documents.

Product status

All products and Services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

Product completeness status

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

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
0800-01	10 December 2025	Non-Confidential	REL 8.0.0 BETA release
0701-08	28 May 2025	Non-Confidential	REL 7.2.2 EAC release
0701-07	29 October 2024	Non-Confidential	REL 7.2.1 EAC release
0701-06	29 March 2024	Non-Confidential	REL 7.2.0 BETA release
0701-05	19 December 2023	Non-Confidential	REL 7.1.4 EAC release
0701-04	7 November 2023	Non-Confidential	REL 7.1.3 EAC release
0701-03	28 September 2023	Non-Confidential	REL 7.1.3 EAC release
0701-02	29 June 2023	Non-Confidential	REL 7.1.2 EAC release
0701-01	28 March 2023	Non-Confidential	REL 7.1.1 BETA-1 release
0302-01	26 July 2022	Non-Confidential	Fifth release

Issue	Date	Confidentiality	Change
0301-01	27 September 2021	Non-Confidential	Fourth release
0100-03	11 May 2021	Non-Confidential	Third release
0100-02	25 March 2021	Non-Confidential	Second release
0100-01	25 November 2020	Non-Confidential	First release

The Change history tables describe the technical changes between released issues of this document in reverse order. Issue numbers match the revision history in [Document release information](#) on page 42.

Table 2: Issue A

Change	Location
First release.	-

Table 3: Differences between Issue A and Issue 0100-02

Change	Location
1. Changed the file name of the component Timer and Watchdog. 2. Added two more components - DMA and Miscellaneous.	See 1.5.1 Test suites on page 9.
Changed the node count in IOVIRT nodes.	See 2.1.4.1 Number of IOVIRT Nodes on page 16.
Added PLATFORM_OVERRIDE_GICITS_ID and PLATFORM_OVERRIDE_GICIRD_LENGTH in the GIC-specific tests section.	See 2.1.5 GIC on page 19.
Removed request_msi, free_msi, its_configure, and get_max_lpi_id APIs in the GIC section.	See 3.1.2 GIC on page 30.
Removed pal_pci_read_config_byte and pci_write_config_byte and added 10 new APIs in the PCIe section.	See 3.1.5 PCIe on page 31.
Removed the create_info_table API in the SMMU section.	See 3.1.6 SMMU on page 32.
Removed pal_pcie.c and pal_pcie_enumeration.c APIs and added 8 new APIs in the Peripheral section.	See 3.1.7 Peripheral on page 33.
Renamed pal_mem_alloc_coherent API to pal_mem_alloc_cacheable API and pal_mem_free_coherent API to pal_mem_free_cacheable API. Added pal_mem_phys_to_virt API in the Miscellaneous section.	See 3.1.13 Miscellaneous on page 36.

Table 4: Differences between Issue 0100-02 and Issue 0100-03

Change	Location
Added get_rp_transaction_frwd_support API in PCIe.	See 3.1.5 PCIe on page 31.
Added pal_is_bdf_exerciser API in Exerciser.	See 3.1.11 Exerciser on page 35.

Table 5: Differences between Issue 0100-03 and Issue 0301-01

Change	Location
Updated the Execution of SBSA ACS	See 2. Execution of SBSA ACS on page 11.
Updated the Porting requirements	See 3. Porting requirements on page 29.
Added memory topic.	See 2.1.8 Memory on page 21.

Change	Location
Updated the SBSA example flow diagram.	See 4.2 SBSA test example flow on page 38.

Table 6: Differences between Issue 0301-01 and Issue 0302-01

Change	Location
Updated information for the PCIe enumeration	See 2. Execution of SBSA ACS on page 11.
Added new APIs in Timer, PCIe, and Miscellaneous	See 3.1.3 Timer on page 30, 3.1.5 PCIe on page 31, and 3.1.13 Miscellaneous on page 36.

Table 7: Differences between Issue 0302-01 and Issue 0701-01

Change	Location
Changes to cp -r FVP/RDN2/platform_name	See 1.5 Steps to customize bare-metal code on page 9
Added MPAM, PMU, RAS and HMAT components	See: <ul style="list-style-type: none"> 2.1.9 MPAM on page 22 2.1.12 PMU on page 26 2.1.11 RAS on page 26 2.1.10 HMAT on page 25
Added PAL APIs	See: <ul style="list-style-type: none"> 3.1.8 MPAM on page 33 3.1.9 RAS on page 34 3.1.13 Miscellaneous on page 36

Table 8: Differences between Issue 0701-01 and Issue 0701-02

Change	Location
Added Named component specific information on Coresight components	See 2.1.4.11 Named component node specific information on page 18
Added PAL APIs	See 3.1.5 PCIe on page 31

Table 9: Differences between Issue 0701-02 and Issue 0701-03

Change	Location
Updated the steps to customize bare-metal code.	See 1.5 Steps to customize bare-metal code on page 9
Removed the API max_pasids from the SMMU APIs table.	See 3.1.6 SMMU on page 32
Updated SBSA example flow diagram in SBSA test example flow.	See 4.2 SBSA test example flow on page 38

Table 10: Differences between Issue 0701-03 and Issue 0701-04

Change	Location
Added new sections, Boot framework, Boot process and boot flow, and Boot framework for Bare-metal.	See, 1.4 Boot framework on page 7, 1.4.1 Boot process and boot flow on page 7, and Boot framework for Bare-metal
Updated the information required for PCIe enumeration.	See, 2.1.2 PCIe on page 13.
Added MMU in abbreviation.	See 1.1 Abbreviations on page 5
Added a new section for MMU configuration.	See, 2.1.13 MMU Configuration on page 27.
Added a new section Peripherals.	See, Peripherals .
Added a new section Bare-metal Boot.	See, Bare-metal Boot .

Change	Location
Added new APIs in Miscellaneous APIs and their details table and updated the Function prototype.	See, 3.1.13 Miscellaneous on page 36.

Table 11: Differences between Issue 0701-04 and Issue 0701-05

Change	Location
Added a new section for Memory map.	See, 3.1.12 Memory map on page 35.
Updated the Miscellaneous APIs and their details table.	See, 3.1.13 Miscellaneous on page 36.
Updated the PCIe APIs and their details table.	See, 2.1.2 PCIe on page 13.

Table 12: Differences between Issue 0701-05 and Issue 0701-06

Change	Location
Updated the folder name from pal_baremetal to baremetal/target/.	See, 1.5 Steps to customize bare-metal code on page 9.
Updated the PCIe root ports and parameters.	See, 2.1.2 PCIe on page 13.
Added new APIs in the Exerciser APIs and their details table.	See, 3.1.11 Exerciser on page 35.

Table 13: Differences between Issue 0701-06 and Issue 0701-07

Change	Location
Updated MPAM, Header file representation.	See, 2.1.9 MPAM on page 22.
Added PCC component.	See, 2.1.9.2 PCC on page 24.
Added a new API in the PCIe APIs and their details table.	See, 3.1.5 PCIe on page 31.
Added a new API in the MPAM APIs and their details table.	See, 3.1.8 MPAM on page 33.

Table 14: Differences between Issue 0701-07 and Issue 0701-08

Change	Location
Updated the steps to customize the bare-metal code.	See, 1.5 Steps to customize bare-metal code on page 9.

Table 15: Differences between Issue 0701-08 and Issue 0800-01

Change	Location
No technical changes.	-

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.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. 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 <u>underline</u>	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: <div>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></div>
SMALL CAPITALS	Terms that have specific technical meanings as defined in the <i>Arm® Glossary</i> . For example, IMPLEMENTATION DEFINED , IMPLEMENTATION SPECIFIC , UNKNOWN , and UNPREDICTABLE .



Caution

We recommend the following. If you do not follow these recommendations your system might not work.



Warning

Your system requires the following. If you do not follow these requirements your system will not work.



Danger

You are at risk of causing permanent damage to your system or your equipment, or of harming yourself.



Note

This information is important and needs your attention.



Tip

This information might help you perform a task in an easier, better, or faster way.



This information reminds you of something important relating to the current content.

Useful resources

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

Arm documents are available on developer.arm.com/documentation.

Confidential documents are only available to licensees, when logged in. Each document link in the following tables provides direct access to the online version of the document.

Arm product resources	Document ID	Confidentiality
Arm® SBSA Architecture Compliance Test Scenario	PJDOC-2042731200-3439	Non-Confidential
Arm® SBSA Architecture Compliance User Guide	101547	Non-Confidential
Arm® SBSA Architecture Compliance Validation Methodology	101544	Non-Confidential
Arm® Server Base System Architecture 7.1	DEN0029H	Non-Confidential