Corstone-1000

Arm Limited

Jan 30, 2026

CONTENTS

CHAPTER

ONE

DISCLAIMER

Arm reference solutions are Arm public example software projects that track and pull upstream components, incorporating their respective security fixes published over time. Arm partners are responsible for ensuring that the components they use contain all the required security fixes, if and when they deploy a product derived from Arm reference solutions.

1.1 Software Architecture

1.1.1 Arm Corstone-1000

Arm Corstone-1000 is a reference solution for IoT devices. It is part of Total Solution for IoT which consists of hardware and software reference implementation.

The combination of Corstone-1000 software and hardware reference solution is PSA Level-2 ready certified as well as Arm SystemReady Devicetree certified.

More information on the Corstone-1000 subsystems product and design can be found on Arm Developer.

This document explicitly focuses on the software part of the solution and provides internal details on the software components. The reference software package of the platform can be retrieved following instructions present in the user guide document.

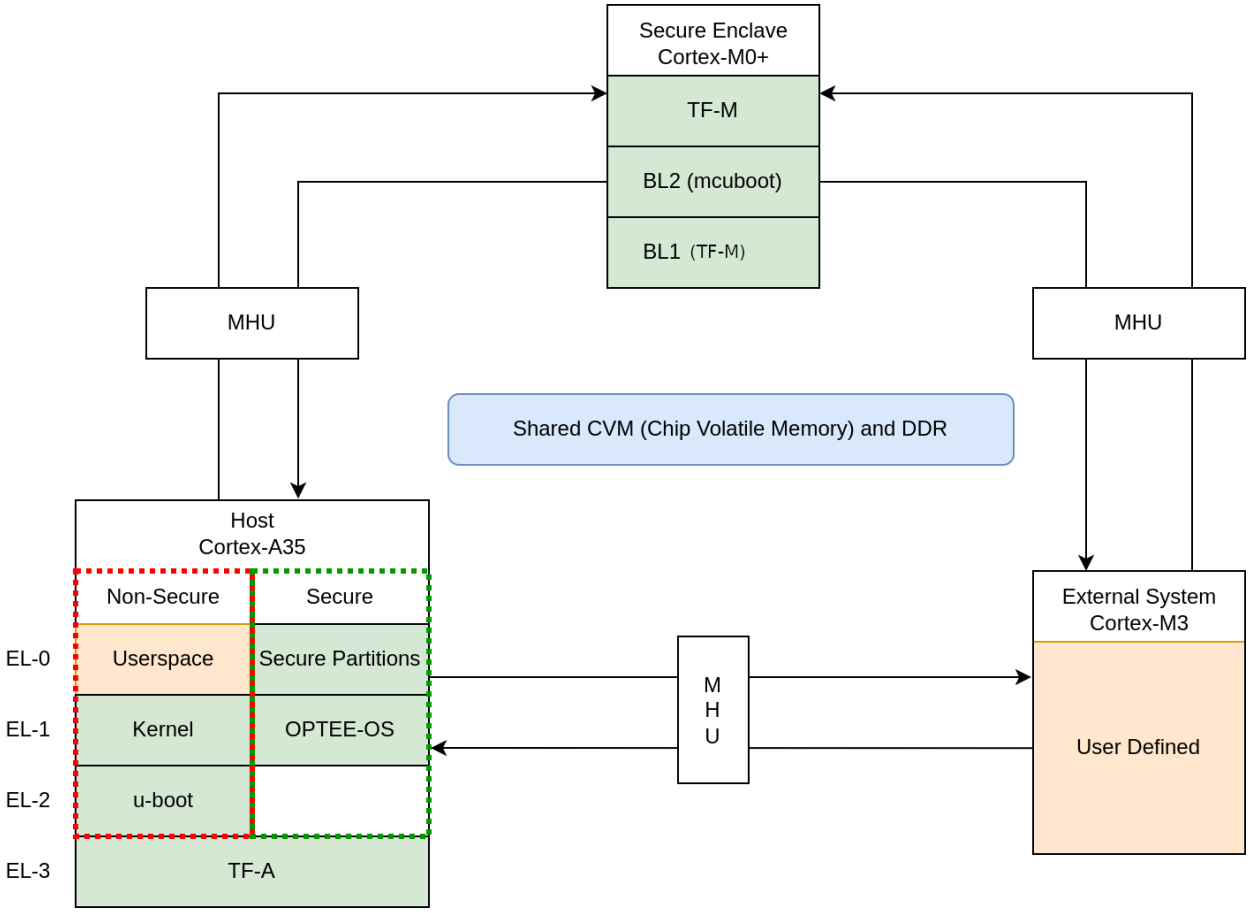
1.1.2 Design Overview

The software architecture of Corstone-1000 platform is a reference implementation of Platform Security Architecture which provides framework to build secure IoT devices.

The base system architecture of the platform is created from three different types of subsystems:

- Secure Enclave
- Host System
- External System

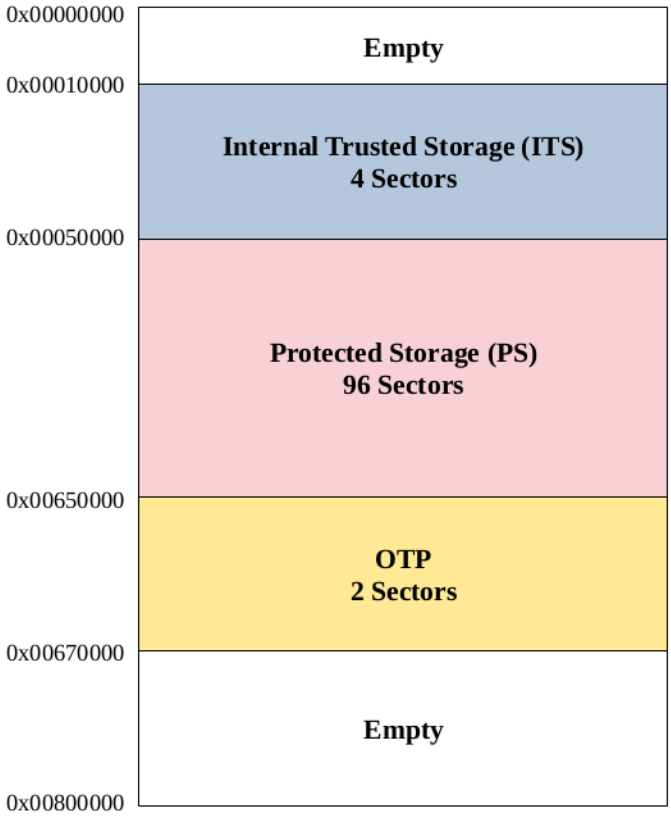
Each subsystem provides different functionality to the overall system on a chip (SoC).

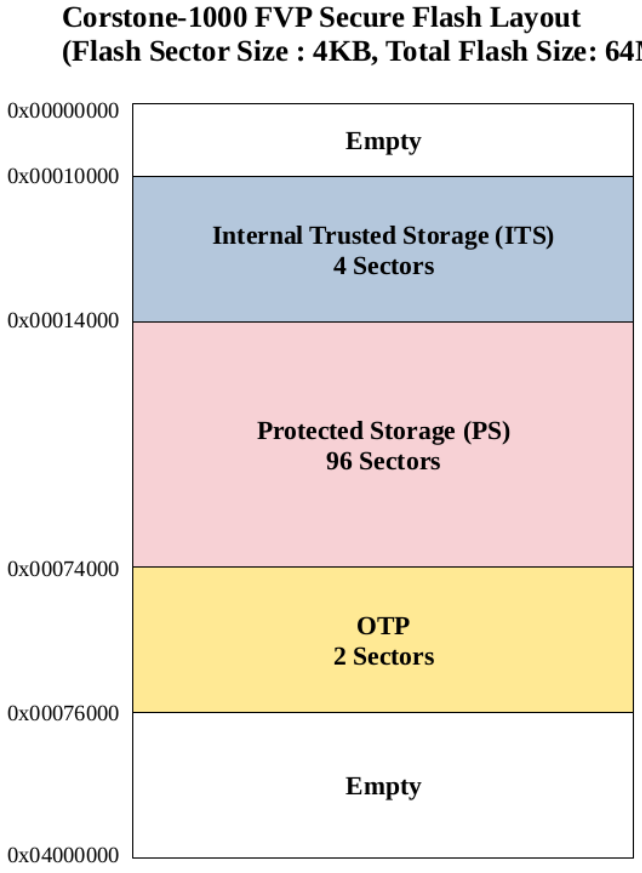


Secure Enclave

The Secure Enclave boots first on system power on, it provides PSA Root of Trust (RoT) and cryptographic functions. It is based on a Cortex-M0+ processor, CC312 Cryptographic Accelerator and peripherals such as watchdog and secure flash.

Corstone-1000 MPS3 Secure Flash Layout
(Flash Sector Size : 64KB, Total Flash Size: 8MB)





Software running on the Secure Enclave is isolated via hardware for enhanced security. Communication with the Secure Enclave is achieved using Message Handling Units (MHUs) and shared memory.

Its software components comprises:

- Trusted Firmware-M (TF-M) BL1
- MCUboot
- TrustedFirmware-M

The software design on the Secure Enclave follows Arm Firmware Framework for M-Profile processor (FF-M) specification.

Host System

The Host System is based on ARM Cortex-A35 processor with standardized peripherals to allow booting a Linux-based operating system (OS). The Cortex-A35 has the TrustZone technology that allows Secure and Non-secure security states in the processor.

The boot process follows Trusted Boot Base Requirements Client. The Host System is taken out of reset by the Secure Enclave system during its final stages of the initialization.

In the Secure world, the Host System runs:

- FF-A Secure Partitions (based on Trusted Services)
- OP-TEE OS

In the Non-secure World, the Host System runs:

- U-Boot
- Linux kernel

The software design in the Host System follows Arm Firmware Framework for Arm A-profile (FF-A) specification. The communication between Non-secure and the Secure world is performed via FF-A messages.

External System

The External System is intended to implement use-case specific functionality.

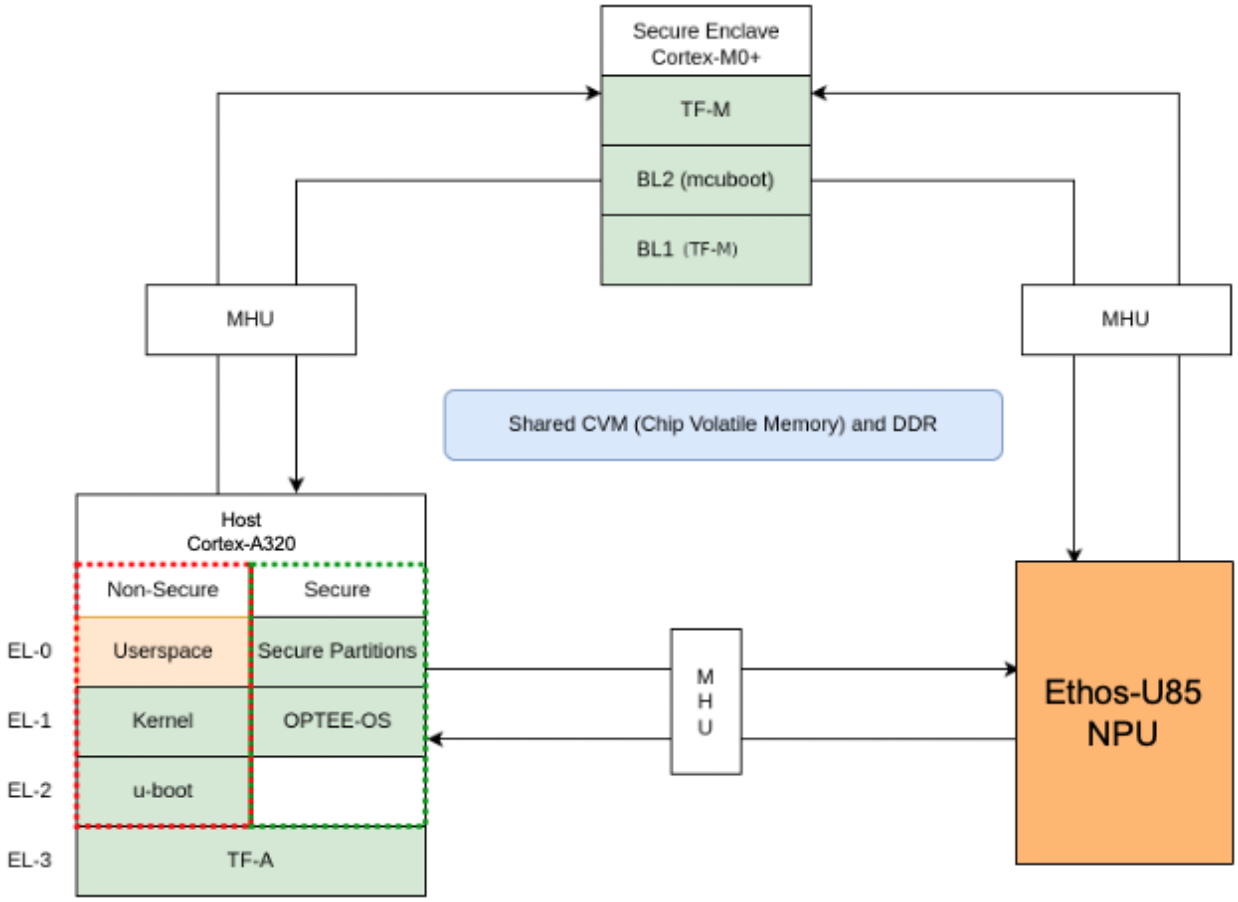
The system is based on Cortex-M3 and runs Keil RTX5.

Communication between the external system and Host (Cortex-A35) can be performed using MHU as transport mechanism. The current software release supports switching the External System ON and OFF.

The Corstone-1000 architecture is designed to cover a range of Power, Performance, and Area (PPA) applications, and enable extension for use-case specific applications, for example, sensors, cloud connectivity, and edge computing.

1.1.3 Corstone-1000 with Cortex-A320 Variant

This variant of the Corstone-1000 platform replaces the Host System's Cortex-A35 processor with a Cortex-A320. In this configuration, the optional External System (previously a Cortex-M3) is replaced by an Arm Ethos-U85 Neural Processing Unit (NPU). The Ethos-U85 runs in the direct drive configuration, where the Host System is responsible for managing the NPU directly.



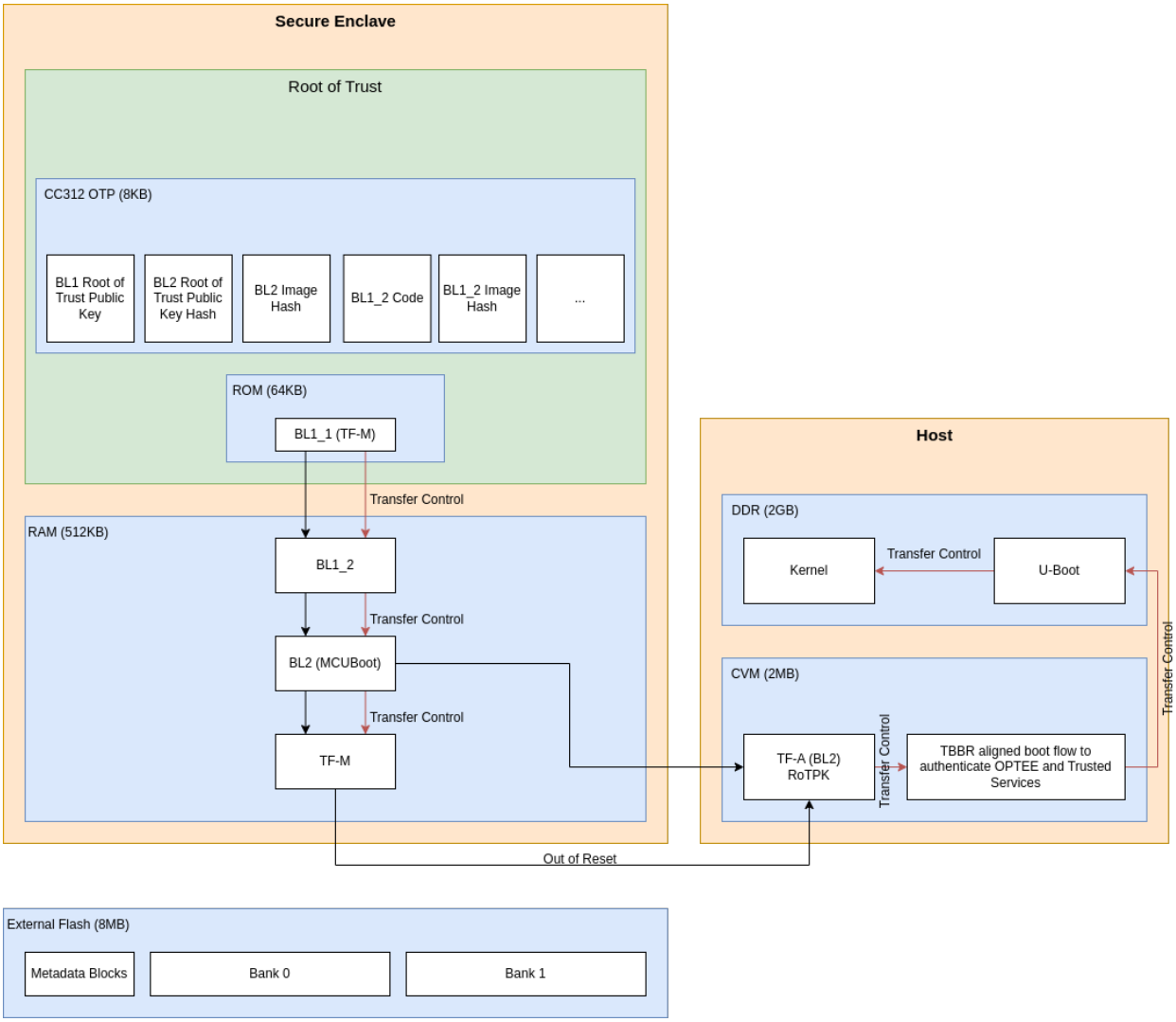
1.1.4 Secure Boot Chain

For the security of a device, it is essential that only authorized software should run on the device.

The Corstone-1000 boot uses a Secure boot chain process where an already authenticated image verifies and loads the following software in the chain.

For the boot chain process to work, the start of the chain should be trusted, forming the Root of Trust (RoT) of the device. The RoT of the device is immutable in nature and encoded into the device by the device manufacturer before it is deployed into the field. In Corstone-1000, the content of the ROM and CC312 One Time Programmable (OTP) memory forms the RoT.

Verification of an image can happen either by comparing the computed and stored hashes, or by checking the signature of the image if the image is signed.



It is a lengthy chain to boot the software on Corstone-1000.

TF-M BL1_1

On power-up, the Secure Enclave begins execution from TF-M BL1_1, which resides in ROM and serves as the Root of Trust (RoT) for the device.

TF-M BL1_1 is the immutable bootloader and is responsible for:

- Provisioning the device during the first boot
- Performing hardware initialization
- Verifying the integrity and authenticity of the next stage in the boot chain

At boot time, TF-M BL1_1:

- Copies the TF-M BL1_2 image from OTP to RAM.
- Verifies the integrity of BL1_2 by comparing its computed hash with the hash stored in OTP.

TF-M BL1_2

During provisioning, the TF-M BL1_2 binary, along with its hashes and cryptographic keys, is stored in One-Time Programmable (OTP) memory.

Once verified, TF-M BL1_2:

- Takes control and verifies the next stage in the boot chain, which is TF-M BL2.
- Computes the hash of the BL2 image and compares it with the BL2 hash stored in OTP to ensure integrity before transferring execution to BL2.

Note: The TF-M BL1 design details can be found in the TF-M design documents.

Important: Corstone-1000 has some differences compared to this design due to memory (OTP/ROM) limitations:

- BL1_1 code size is larger than needed because it handles most of the hardware initialization instead of the BL1_2.
- BL1_2 cannot be updated during provisioning time because the provisioning bundle that contains its code is located in the ROM.
- BL1_2 does not use the post-quantum LMS verification.
- BL2 cannot be updated because it is verified by comparing the computed hash to the hash stored in the OTP.

TF-M BL2

In this system, TF-M BL2 refers to MCUBoot.

On the first boot, MCUBoot can provision additional cryptographic keys. It is responsible for authenticating both:

- TF-M (Trusted Firmware-M), and
- The initial bootloader of the Host system, Trusted Firmware-A (TF-A) BL2

This authentication is done by verifying the digital signatures of the respective images.

MCUBoot performs image verification in the following steps:

1. Load the image from non-volatile memory into RAM.
2. Validate the image's signature using the corresponding public key.

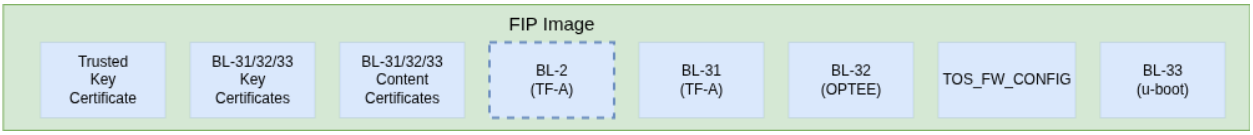
Note: The public key present in the image header is validated by comparing with the hash. Depending on the image, the hash of the public key is either stored in the OTP or part of the software which is being already verified in the previous stages.

The execution control is passed to TF-M after the verification. As the runtime executable of the Secure Enclave, TF-M initializes itself before bringing the Host system out of reset.

Host System Authentication

The Host system follows the boot standard defined in the Trusted Board Boot Requirements Client to authenticate the Secure and Non-secure software.

The Firmware Image Package (FIP) packs bootloader images and other payloads into a single archive.



The FIP for Corstone-1000 contains:

- Trusted firmware-A BL2
- AP EL3 Runtime firmware, BL31 image
- AP Secure Payload, BL32 image
- AP Normal world firmware -U-boot, BL33 image
- Trusted OS Firmware configuration file used by Trusted OS (BL32), TOS_FW_CONFIG
- Key certificates
- Content certificates

To load and validate TF-A BL2, TF-M BL2 first parses the GUID Partition Table (GPT) to locate the FIP. It then determines the offset of TF-A BL2 within the FIP.

Note: TF-M does not check the FIP signature, it only checks the TF-A BL2's signature in the FIP.

Important: The implicitly trusted components are:

- A SHA-256 hash of the Root of Trust Public Key (ROTPK) - For development purposes, a development ROTPK is used and its hash embedded into the TF-A BL2 image. This public key is provided by the TF-A source code.
- TF-A BL2 image - it can be trusted because it has been verified by TF-M BL2 before starting TF-A.

The remaining components in the Chain of Trust (CoT) are either certificates or bootloader images.

Bootloader Authentication

The FIP contains two types of certificates:

- **Content Certificates** - used to store the hash of a bootloader image.
- **Key Certificates** - used to verify public keys used to sign Content Certificates.

The Host system bootloader images are authenticated by computing their hash and comparing it to the corresponding hash found in the Content Certificate.

Certificates Verification

The public keys defined in the Trusted Key Certificate are used to verify the later certificates in the CoT process. The Trusted Key Certificate is verified with the Root of Trust Public Key.

UEFI Authenticated Variables

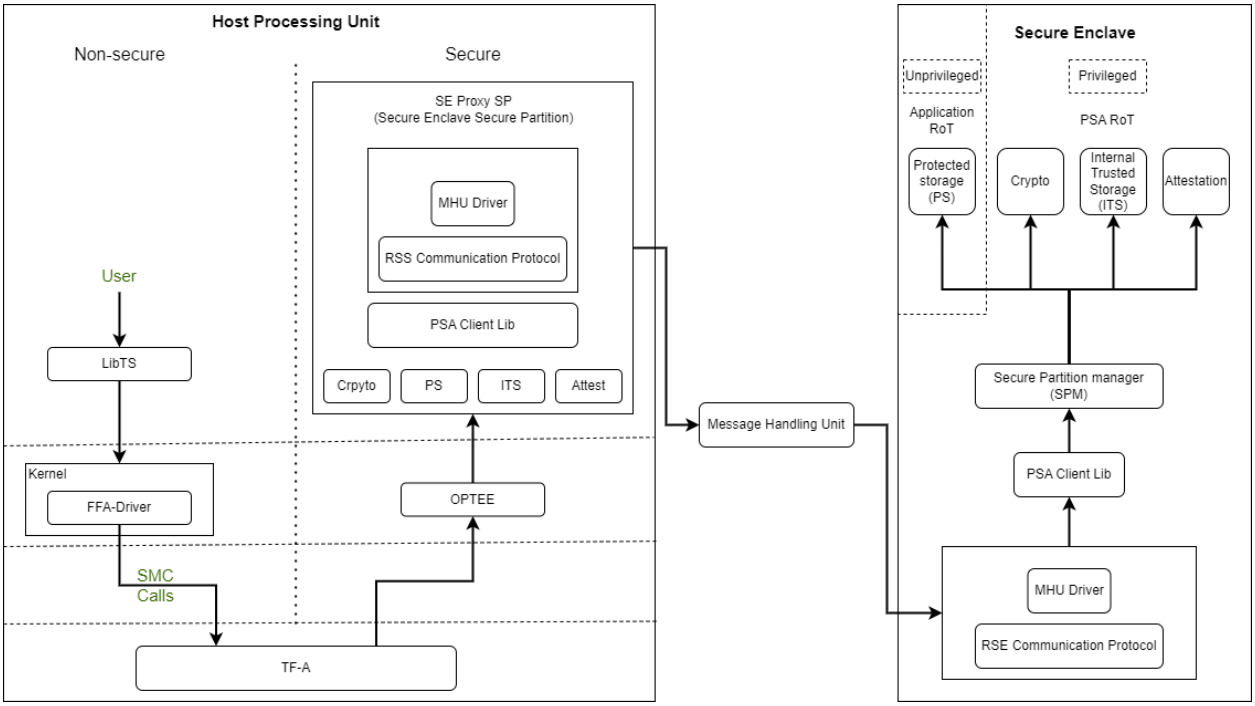
For UEFI Secure Boot, authenticated variables can be accessed from the secure flash. The feature has been integrated in U-Boot, which authenticates the images as per the UEFI specification before executing them.

1.1.5 Secure Services

Corstone-1000 is unique in offering a secure environment for running trusted workloads. While the Host system includes TrustZone technology, the platform also features a hardware-isolated Secure Enclave, specifically designed to execute these secure workloads.

In Corstone-1000, essential Secure Services—such as Cryptography, Protected Storage, Internal Trusted Storage, and Attestation—are provided through PSA Functional APIs implemented in TF-M.

From the user's perspective, there is no difference when communicating with these services, whether they run in the Secure Enclave or in the Secure world of the Host system. The diagram below illustrates the data flow for such calls.



The Secure Enclave Proxy Secure Partition (SE Proxy SP) is a proxy managed by OP-TEE that forwards Secure Service calls to the Secure Enclave. This communication uses the RSE communication protocol. While the protocol supports shared memory and MHU interrupts as a doorbell mechanism between cores, in Corstone-1000, the entire message is currently transmitted through the MHU channels. Corstone-1000 implements Isolation Level 2 using the Cortex-M0+ Memory Protection Unit (MPU).

Users can define their own secure services to run either in the Host system's Secure World or in the Secure Enclave. This choice involves a trade-off between latency and security. Services running in the Secure Enclave benefit from strong, hardware-enforced isolation, offering higher security but at the cost of increased latency. In contrast, services

running in the Host Secure World experience lower latency, but rely on TrustZone technology for virtualized isolation, which offers comparatively less robust security.

1.1.6 PSA Secure Firmware Update

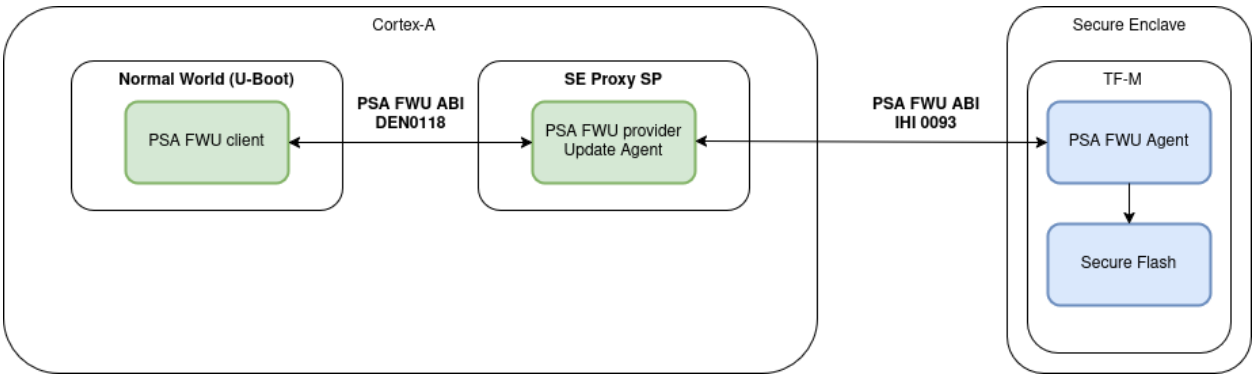
The Arm Corstone-1000 platform necessitates a robust, secure, and flexible firmware update mechanism including partial capsule update to ensure fielded devices can receive critical patches, feature enhancements, and security fixes without compromising system integrity. To meet these requirements, we have implemented the Platform Security Architecture (PSA) Firmware Update (FWU) framework on Corstone-1000, leveraging Trusted Firmware-M (TF-M) for the Secure Enclave, U-Boot as the host-side client on Cortex-A, and the UEFI capsule update mechanism for payload encapsulation. This design supports both the Fixed Virtual Platform (FVP) and the Field Programmable Gate Array (FPGA) targets, providing consistent behavior across simulation and silicon-based deployments. The Corstone-1000 supports FWU which complies with the Platform Security Firmware Update for the A-profile Arm Architecture and PSA Firmware Update IHI 0093 specifications.

To standardize and streamline capsule creation with multiple FMP payloads, the EDK2 capsule generation tool tool has been integrated into the meta-arm Yocto layer for Corstone-1000. This integration involves defining build rules for generating UEFI capsules as part of the firmware image build process. Configuration parameters exposed in the recipe allow developers to specify the number of FMP payloads, target image GUIDs, version numbers etc. This capsule ensures that all update payloads conform to the UEFI FMP specification and are ready for validation and delivery by U-Boot.

The FWU solution for Corstone-1000 is composed of three primary domains:

- Host System
- Trusted Services intermediary
- Secure Enclave

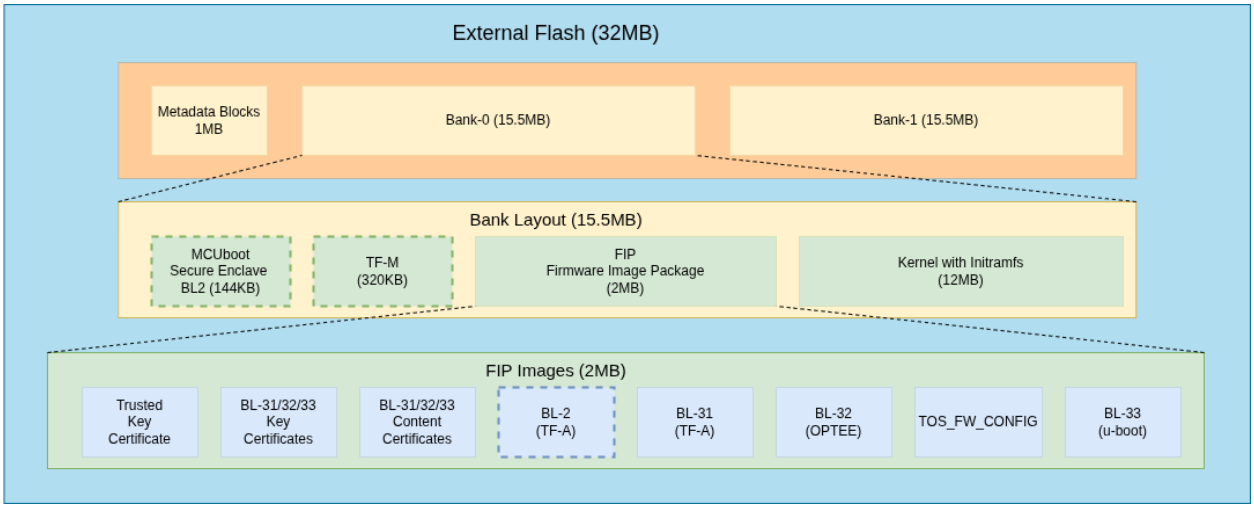
Each domain has distinct responsibilities and communicates through standardized interfaces.



On the host side, U-Boot functions as the FWU client and orchestrates the update process from capsule retrieval to payload delivery based on PSA FWU DEN0018 specification via Arm FF-A framework. The Trusted-Services SE Proxy secure partition serves as a gateway between the non-secure host environment and the Secure Enclave. The PSA FWU service running in the Trusted Services implementation forwards the data to the Secure Enclave via MHU-based PSA calls. Within the Secure Enclave, the PSA FWU Agent, conforming to PSA Firmware Update IHI 0093 specification, orchestrates the actual flash programming, metadata management, and rollback protection mechanisms. The agent relies on a bespoke shim layer to abstract hardware-specific flash operations and bootloader interactions.

As defined in the specification, the external flash is divided into two banks: one bank holds the currently running images, while the other is used to stage new images.

There are four updatable components: **BL2**, **TF-M**, **the FIP** and **the Kernel Image** (the initramfs bundle). New images are delivered and accepted in the form of UEFI capsules.



When a FWU is initiated on Corstone-1000, the following sequence of operations takes place:

1. **Capsule Retrieval and Preparation**

   U-Boot on the host system retrieves the firmware capsule. It validates the capsule header and parses the FMP (Firmware Management Protocol) descriptor list to identify the payloads to be updated.

   For each FMP descriptor, U-Boot:

   Splits the firmware payload into 4 KiB chunks. Invokes the PSA_FWU_Update API for each chunk, transmitting the buffer address via the FF-A (Firmware Framework for Arm) shared memory interface.

2. **Secure Transmission and Forwarding**

   The PSA Firmware Update (FWU) service, running as part of Trusted Services, receives the chunks through Secure Partition Client (SPC) calls. It forwards these chunks to the Secure Enclave using MHU-based PSA calls.

3. **Flashing Within the Secure Enclave**

   Inside the Secure Enclave, the PSA FWU Agent dispatches each chunk to the shim layer.

   The shim layer:

   Erases the corresponding sectors in the non-active flash bank. Writes the received firmware chunks at the correct offsets. During partial updates, it also copies static partitions from the active bank to the non-active one to maintain consistency.

4. **Finalization and Boot Preparation**

   After all chunks are successfully written:

   The shim updates the firmware manifest and the EFI System Resource Table (ESRT) entries to reflect the new image version. This step enables the bootloader to recognize the new firmware for a trial boot. The platform then performs an automatic reset, booting into the non-active bank in trial mode.

5. **Trial Boot and Confirmation**

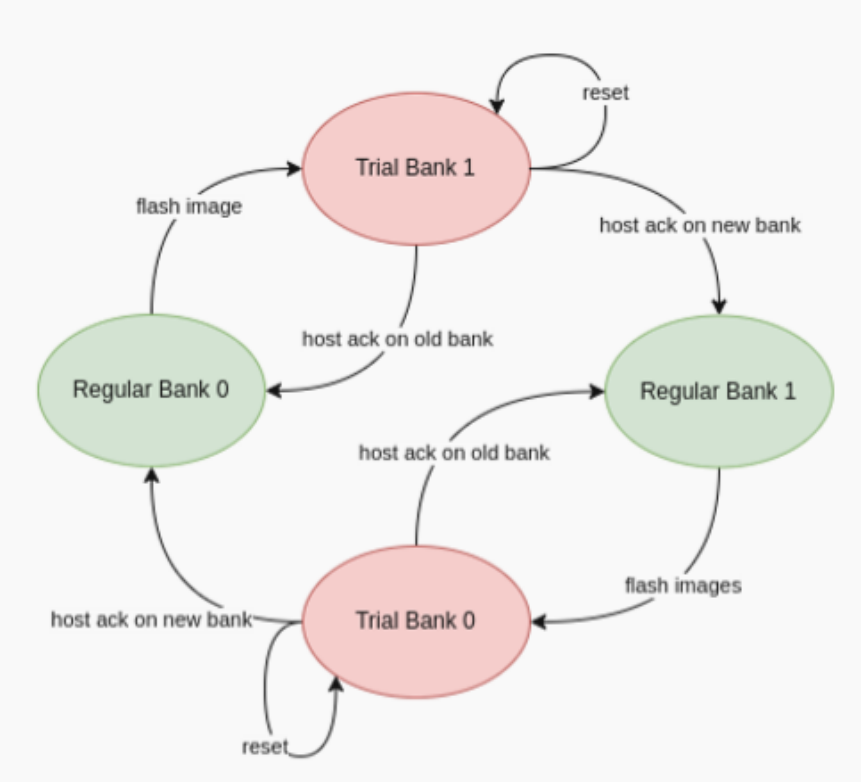
   In trial mode, U-Boot evaluates the new firmware and issues either an accept or reject command using the PSA FWU ABI. These commands are sent to the Secure Enclave, instructing the shim to update the firmware metadata accordingly.

6. **Recovery and Fallback Mechanism**

   If the trial boot is successful, the host sends an acknowledgment, transitioning the firmware state from ‘trial’ to ‘regular’.

   If the system fails or becomes unresponsive:

A watchdog timer triggers a system reset. The BL1 firmware in the Secure Enclave detects repeated failures and reverts to the previously known-good flash bank. This rollback mechanism ensures the device remains operational and recoverable, even after a failed update.

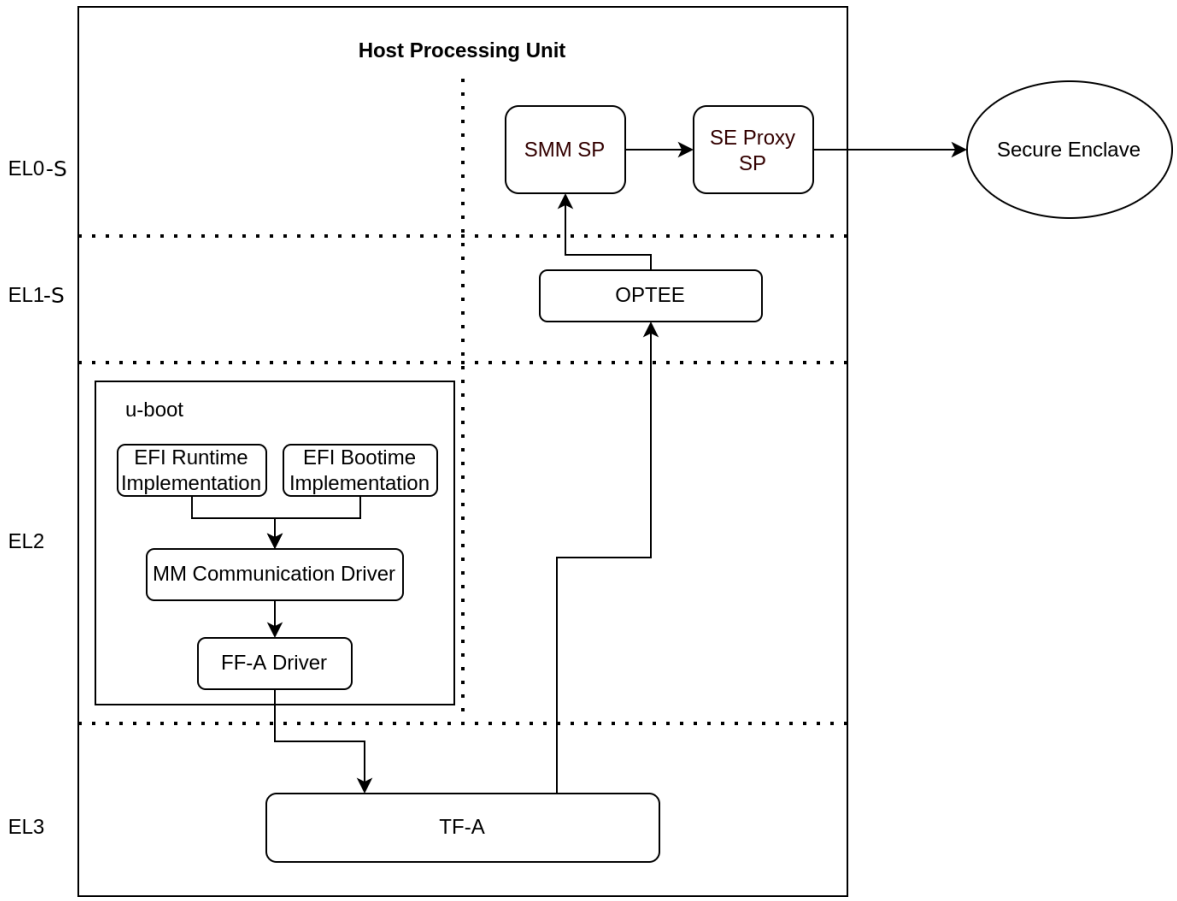


1.1.7 UEFI Runtime Support in U-Boot

The implementation of UEFI boot-time and runtime APIs requires persistent variable storage. In Corstone-1000, UEFI variables are stored using the Protected Storage (PS) service.

The diagram below illustrates the data flow for storing UEFI variables. U-Boot's UEFI subsystem communicates with the Secure World using the U-Boot FF-A driver, which interfaces with the UEFI System Management Mode (SMM) service.

The SMM service provides support for the UEFI System Management Mode. This support is implemented by the SMM Gateway secure partition. The SMM service then uses the Proxy Protected Storage (PS) provided by the SE Proxy SP. These PS calls are forwarded to the Secure Enclave, following the communication path described earlier.



1.1.8 References

- Arm Developer
- Arm Security Architectures

1.2 Build, Flash and Run

1.2.1 Notice

The Corstone-1000 software stack uses the Yocto Project to build a tiny Linux distribution suitable for the Corstone-1000 platform (kernel and initramfs filesystem less than 5 MB on the flash). The Yocto Project relies on the BitBake tool as its build tool. Please see Yocto Project documentation for more information.

1.2.2 Prerequisites

This guide assumes that your host machine is running Ubuntu 24.04 LTS (with `sudo` rights), with at least 32GB of free disk space and 16GB of RAM as minimum requirement.

The following prerequisites must be available on the host system:

- Git 2.39.2 or greater.
- Python 3.11.2 or greater.
- GNU Tar 1.34 or greater.
- GNU Compiler Collection 12.2 or greater.
- GNU Make 4.3 or greater.
- tmux 3.3 or greater.

Please follow the steps described in the Yocto mega manual:

- Compatible Linux Distribution
- Build Host Packages

1.2.3 Targets

The Corstone-1000 software stack can be run on:

- Arm Corstone-1000 Ecosystem FVP (Fixed Virtual Platform)
- Arm Corstone-1000 for MPS3

> **Important:** Arm Corstone-1000 for MPS3 requires an additional 32 MB QSPI flash PMOD module. For more information see the Application Note AN550 document.

1.2.4 Yocto Stable Branch

Corstone-1000 software stack is built on top of Yocto Project's Whinlatter release.

1.2.5 Software Components

Within the Yocto Project, each component included in the Corstone-1000 software stack is specified as a BitBake recipe. The recipes specific to the Corstone-1000 BSP are located at: `${WORKSPACE}/meta-arm/meta-arm-bsp/`.

Important: `${WORKSPACE}` refers to the absolute path to your workspace where the *meta-arm* repository will be cloned.

Consider exporting it (e.g., `export WORKSPACE=$(realpath .)`) if you're already in the workspace directory, so you can copy and paste the commands from this guide verbatim.

Important: `${TARGET}` is either `mps3` or `fvp`.

Consider exporting it (e.g., `export TARGET=mps3`) so you can copy and paste the commands from this guide verbatim.

The Yocto machine config files for the Corstone-1000 FVP and MPS3 targets are:

- `${WORKSPACE}/meta-arm/meta-arm-bsp/conf/machine/include/corstone1000.inc`
- `${WORKSPACE}/meta-arm/meta-arm-bsp/conf/machine/corstone1000-${TARGET}.conf`

Note: All the paths stated in this document are absolute paths.

Host Processor Components

Trusted Firmware-A

bbappend	`${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-bsp/trusted-firmware-a/trusted-firmware-a_%.bbappend`
Recipe	`${WORKSPACE}/meta-arm/meta-arm/recipes-bsp/trusted-firmware-a/trusted-firmware-a_2.13.0.bb`

Trusted Services

bbappend	`${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-security/trusted-services/libts_%.bbappend`
bbappend	`${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-security/trusted-services/ts-psa-crypto-api-test_%.bbappend`
bbappend	`${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-security/trusted-services/ts-psa-iat-api-test_%.bbappend`
bbappend	`${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-security/trusted-services/ts-psa-its-api-test_%.bbappend`
bbappend	`${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-security/trusted-services/ts-psa-ps-api-test_%.bbappend`
bbappend	`${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-security/trusted-services/ts-sp-se-proxy_%.bbappend`
bbappend	`${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-security/trusted-services/ts-sp-smm-gateway_%.bbappend`
Recipe	`${WORKSPACE}/meta-arm/meta-arm/recipes-security/trusted-services/libts_git.bb`
Recipe	`${WORKSPACE}/meta-arm/meta-arm/recipes-security/trusted-services/ts-psa-crypto-api-test_git.bb`
Recipe	`${WORKSPACE}/meta-arm/meta-arm/recipes-security/trusted-services/ts-psa-iat-api-test_git.bb`
Recipe	`${WORKSPACE}/meta-arm/meta-arm/recipes-security/trusted-services/ts-psa-its-api-test_git.bb`
Recipe	`${WORKSPACE}/meta-arm/meta-arm/recipes-security/trusted-services/ts-psa-ps-api-test_git.bb`
Recipe	`${WORKSPACE}/meta-arm/meta-arm/recipes-security/trusted-services/ts-sp-smm-gateway_git.bb`
Recipe	`${WORKSPACE}/meta-arm/meta-arm/recipes-security/trusted-services/ts-sp-se-proxy_git.bb`

OP-TEE

bbappend	${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-security/optee/optee-os_4.%.bbappend
Recipe	${WORKSPACE}/meta-arm/meta-arm/recipes-security/optee/optee-os_4.7.0.bb

U-Boot

bbappend	${WORKSPACE}/meta-arm/meta-arm/recipes-bsp/u-boot/u-boot_%.bbappend
bbappend	${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-bsp/u-boot/u-boot_%.bbappend
Recipe	${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-bsp/u-boot/u-boot_2025.04.bb

Linux

The distribution is based on the Poky distribution which is a Linux distribution stripped down to a minimal configuration.

The provided distribution is based on BusyBox and built using musl libc.

bbappend	${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-kernel/linux/linux-yocto_%.bbappend
Recipe	${WORKSPACE}/core/meta/recipes-kernel/linux/linux-yocto_6.12.bb
defconfig	${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-kernel/linux/files/corstone1000/defconfig

Secure Enclave Components

Trusted Firmware-M

bbappend	${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-bsp/trusted-firmware-m/trusted-firmware-m_%.bbappend
Recipe	${WORKSPACE}/meta-arm/meta-arm/recipes-bsp/trusted-firmware-m/trusted-firmware-m_2.2.1.bb

External System Processor Components

RTX Real-Time operating system

An example application that uses the RTX Real-Time Operating System.

The application project can be found here.

Recipe	${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-bsp/external-system/external-system_0.1.0.bb

1.2.6 Build

Warning: Building binaries natively on Windows and AArch64 Linux is not supported.

Use an Intel or AMD 64-bit architecture Linux based development machine to build the software stack and transfer the binaries to run the software stack on an FVP in Windows or AArch64 Linux if required.

1. Create a new folder that will be your workspace.

```
mkdir ${WORKSPACE}
cd ${WORKSPACE}
```

2. Install kas version 4.4 with sudo rights.

```
sudo pip3 install kas==4.4
```

   Ensure the kas installation directory is visible on the `$PATH` environment variable.

3. Clone the *meta-arm* Yocto layer in the workspace `${WORKSPACE}`.

```
cd ${WORKSPACE}
git clone https://git.yoctoproject.org/git/meta-arm -b CORSTONE1000-2025.12
```

4. Build a Corstone-1000 image:

```
kas build meta-arm/kas/corstone1000-${TARGET}.yml:meta-arm/ci/debug.yml
```

Important: Accept the EULA at https://developer.arm.com/downloads/-/arm-ecosystem-fvps/eula to build a Corstone-1000 image for FVP as follows:

```
export ARM_FVP_EULA_ACCEPT="True"
```

Warning: **The External System Processor is not available on the Corstone-1000 with Cortex-A320 FVP.**

Access to the External System Processor is disabled by default on **Corstone-1000 with Cortex-A35**.

To build the Corstone-1000 image with External System Processor enabled, run:

```
kas build meta-arm/kas/corstone1000-${TARGET}.yml:meta-arm/ci/debug.
↪yml:meta-arm/kas/corstone1000-extsys.yml
```

Warning: **The Ethos-U85 Neural Processing Unit (NPU) is only available on the Corstone-1000 with Cortex-A320 FVP.**

To build the Corstone-1000 image with the Ethos-U85 NPU enabled, run:

```
kas build meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml:meta-
↪arm/kas/corstone1000-a320.yml
```

A clean build takes a significant amount of time given that all of the development machine utilities are also built along with the target images. Those development machine utilities include executables (Python, CMake, etc.) and the required toolchains.

Once the build succeeds, all output binaries will be placed in `${WORKSPACE}/build/tmp/deploy/images/corstone1000-${TARGET}/`

Everything apart from the Secure Enclave ROM firmware and External System firmware, is bundled into a single binary, the `corstone1000-flash-firmware-image-corstone1000-${TARGET}.wic` file.

The output binaries run in the Corstone-1000 platform are the following:

- The Secure Enclave ROM firmware: `${WORKSPACE}/build/tmp/deploy/images/corstone1000-${TARGET}/bl1.bin`
- The External System Processor firmware: `${WORKSPACE}/build/tmp/deploy/images/corstone1000-${TARGET}/es_flashfw.bin`
- The internal firmware flash image: `${WORKSPACE}/build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000-flash-firmware-image-corstone1000-${TARGET}.wic`

1.2.7 Flash

Note: The steps below only apply to the MPS3. The FVP being a software application running on your development machine does not require any firmware flashing. Refer to *this* section for running the software stack on FVP.

Important: When preparing the SD card for flashing, ensure that it is at least 4GB in size and formatted with a FAT32 partition. Using smaller cards or unsupported file systems (e.g., exFAT, NTFS) may cause the flashing process to fail or the device to become unresponsive.

1. Download the FPGA bit file image `AN550: Arm® Corstone™-1000 for MPS3 Version 2.0` on the Arm Developer website. Click on the `Download AN550 bundle` button and login to download the file.

   The directory structure of the FPGA bundle is as shown below:

```
Boardfiles
├── config.txt
├── MB
│   ├── BRD_LOG.TXT
│   ├── HBI0309B
│   │   ├── AN550
│   │   │   ├── AN550_v2.bit
│   │   │   ├── an550_v2.txt
│   │   │   └── images.txt
│   │   ├── board.txt
│   │   └── mbb_v210.ebf
│   └── HBI0309C
│       ├── AN550
│       │   ├── AN550_v2.bit
│       │   ├── an550_v2.txt
│       │   └── images.txt
│       ├── board.txt
```

(continues on next page)

(continued from previous page)

```
│       └── mbb_v210.ebf
└── SOFTWARE
        ├── an550_st.axf
        ├── bl1.bin
        ├── cs1000.bin
        └── ES0.bin
```

2. Depending upon the MPS3 board version, you should update the `images.txt` file (found in the corresponding `HBI0309x` folder e.g. `Boardfiles/MB/HBI0309${BOARD_VERSION}/AN550/images.txt`) so it points to the images under the `SOFTWARE` directory. Where `${BOARD_VERSION}` is a variable containing the board printed on the MPS3 board.

   The `images.txt` file compatible with the latest version of the software stack can be seen below;

```
;*************************************************
;       Preload port mapping                     *
;*************************************************
;  PORT 0 & ADDRESS: 0x00_0000_0000 QSPI Flash (XNVM) (32MB)
;  PORT 0 & ADDRESS: 0x00_8000_0000 OCVM (DDR4 2GB)
;  PORT 1        Secure Enclave (M0+) ROM (64KB)
;  PORT 2        External System 0 (M3) Code RAM (256KB)
;  PORT 3        Secure Enclave OTP memory (8KB)
;  PORT 4        CVM (4MB)
;*************************************************

[IMAGES]
TOTALIMAGES: 3      ;Number of Images (Max: 32)

IMAGE0PORT: 1
IMAGE0ADDRESS: 0x00_0000_0000
IMAGE0UPDATE: RAM
IMAGE0FILE: \SOFTWARE\bl1.bin

IMAGE1PORT: 0
IMAGE1ADDRESS: 0x00_0000_0000
IMAGE1UPDATE: AUTOQSPI
IMAGE1FILE: \SOFTWARE\cs1000.bin

IMAGE2PORT: 2
IMAGE2ADDRESS: 0x00_0000_0000
IMAGE2UPDATE: RAM
IMAGE2FILE: \SOFTWARE\es0.bin
```

3. Copy `bl1.bin` from `${WORKSPACE}/build/tmp/deploy/images/corstone1000-mps3` to the `SOFTWARE` directory of the FPGA bundle.

4. Copy `es_flashfw.bin` from `${WORKSPACE}/build/tmp/deploy/images/corstone1000-mps3` to the `SOFTWARE` directory of the FPGA bundle and rename the binary to `es0.bin`.

5. Copy `corstone1000-flash-firmware-image-corstone1000-mps3.wic` from `${WORKSPACE}/build/tmp/deploy/images/corstone1000-mps3` to the `SOFTWARE` directory of the FPGA bundle and rename the wic image to `cs1000.bin`.

Note: Renaming of the images is required because the MCC firmware has a limit of 8 characters for file name and 3

characters for file extension.

After making all modifications above, copy the FPGA bit file bundle to the board's SDCard and reboot the MPS3.

1.2.8 Run

Once the target is turned ON, the Secure Enclave will start to boot, wherein the relevant memory contents of the `*.wic` file are copied to their respective memory locations. Firewall policies are enforced on memories and peripherals before bringing the Host Processor out of reset.

The Host Processor will boot TrustedFirmware-A, OP-TEE, U-Boot and then Linux before presenting a login prompt.

MPS3

1. Open 4 serial port comms terminals on the host machine. Those might be `ttyUSB0`, `ttyUSB1`, `ttyUSB2`, and `ttyUSB3` on Linux machines.

- `ttyUSB0` for MCC, OP-TEE and Secure Partition
- `ttyUSB1` for Secure Enclave (Cortex-M0+)
- `ttyUSB2` for Host Processor (Cortex-A35)
- `ttyUSB3` for External System Processor (Cortex-M3)

   The serial ports might be different on Windows machines.

   Run the following commands in separate terminal instances on Linux:

```
sudo picocom -b 115200 /dev/ttyUSB0
```

```
sudo picocom -b 115200 /dev/ttyUSB1
```

```
sudo picocom -b 115200 /dev/ttyUSB2
```

```
sudo picocom -b 115200 /dev/ttyUSB3
```

   Important: Plug a connected Ethernet cable to the MPS3 or it will wait for a network connection for a considerable amount of time, printing the following on the Host Processor terminal (`ttyUSB2`):

```
Generic PHY 40100000.ethernet-ffffffff:01: attached PHY driver (mii_bus:phy_
↪addr=40100000.ethernet-ffffffff:01, irq=POLL)
smsc911x 40100000.ethernet eth0: SMSC911x/921x identified at 0xffffffc008e50000,␣
↪IRQ: 17
Waiting up to 100 more seconds for network.
```

2. Once the system boot is completed, you should see console logs on the serial port terminals. Once the Host Processor is booted completely, user can login to the shell using `root` login.

   Important: The secure flash might be completely filled if the system does not boot and only the Secure Enclave logs (`ttyUSB1`) are visible.

Clean the secure flash if that is the case following the steps *here*.

FVP

A Fixed Virtual Platform (FVP) model of the Corstone-1000 platform must be available to run the Corstone-1000 FVP software image.

A Yocto recipe is provided to download the latest supported FVP version.

The recipe is located at `${WORKSPACE}/meta-arm/meta-arm/recipes-devtools/fvp/fvp-corstone1000.bb`. This recipe supports selecting different Corstone-1000 FVP models via MACHINE_FEATURES:

- `cortexa320` use the Cortex-A320 Host Processor with Ethos U85 enabled FVP build
- (default) use the Cortex-A35 Host Processor with Cortex-M3 External System FVP build

The latest FVP version is `11.23.25` for Corstone-1000 with Cortex-A35 and `11.30.27` for Corstone-1000 with Cortex-A320, and each model is automatically downloaded and installed when using the `runfvp` command as detailed below.

Note:

```
kas shell meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml \
-c "../meta-arm/scripts/runfvp -- --version"
```

The FVP can also be manually downloaded from Arm Developer to download the Corstone-1000 platform FVP installer. Follow the instructions of the installer to setup the FVP.

1. Run `tmux`:

   ```
   cd ${WORKSPACE} && tmux
   ```

2. Run the FVP within `tmux`:

   ```
   kas shell meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml \
   -c "../meta-arm/scripts/runfvp --terminals=tmux"
   ```

   When the script is executed, three terminal instances will be launched:

   - one for the Secure Enclave processing element
   - two for the Host processor processing element.

   ```
   corstone1000-fvp login:
   ```

3. Login using the `root` username.

1.2.9 Security Issue Reporting

To report any security issues identified with Corstone-1000, please send an email to psirt@arm.com.

1.3 Tests

Important: All the tests below assume you have already built the software stack at least once following the instructions *here*.

1.3.1 Reports

Reports for the tests conducted on the Corstone-1000 software (CORSTONE1000-2025.12) release are available for reference here.

1.3.2 Clean Secure Flash

Important: The MPS3 secure flash needs to be cleared before running tests. This is to erase the flash cleanly and prepare a clean board environment for testing.

1. Clone the *systemready-patch* repository to your ${WORKSPACE}.

```
cd ${WORKSPACE}
git clone https://git.gitlab.arm.com/arm-reference-solutions/systemready-
↪patch.git -b CORSTONE1000-2025.12
```

2. Copy the secure flash cleaning Git patch file to your copy of *meta-arm*.

```
cp -f systemready-patch/embedded-a/corstone1000/erase_flash/0001-embedded-a-
↪corstone1000-clean-secure-flash.patch meta-arm
```

3. Apply the Git patch to *meta-arm*.

```
cd meta-arm
git apply 0001-embedded-a-corstone1000-clean-secure-flash.patch
```

4. Rebuild the software stack.

```
cd ${WORKSPACE}
kas shell meta-arm/kas/corstone1000-mps3.yml:meta-arm/ci/debug.yml
bitbake -c cleansstate trusted-firmware-m corstone1000-flash-firmware-image
bitbake -c build corstone1000-flash-firmware-image
```

5. Replace the `bl1.bin` file on the SD card with `${WORKSPACE}/build/tmp/deploy/images/corstone1000-mps3/bl1.bin`.

6. Reboot the board to completely erase the secure flash.

   The following message log from TrustedFirmware-M should be displayed on the Secure Enclave terminal (`ttyUSB1`):

```
!!!SECURE FLASH HAS BEEN CLEANED!!!
NOW YOU CAN FLASH THE ACTUAL CORSTONE1000 IMAGE
PLEASE REMOVE THE LATEST ERASE SECURE FLASH PATCH AND BUILD THE IMAGE AGAIN
```

7. Whilst still in the `kas` shell, revert the changes the patch introduced by running the following commands:

```
cd ${WORKSPACE}/meta-arm
git reset --hard
cd ..
bitbake -c cleansstate trusted-firmware-m corstone1000-flash-firmware-image
exit
```

8. Follow the *instructions* to build a clean software stack and flash the MPS3 with it.

You can proceed with the test instructions in the following section after having done all the above.

1.3.3 SystemReady IR

Important: **SystemReady IR** has now been renamed **SystemReady Devicetree Band**.

Running the tests described below requires USB drives. In our testing, not all USB drive models worked well with the MPS3.

Here are the USB drive models that were stable in our test environment:

- HP v165w 8 GB USB Flash Drive
- SanDisk Ultra 32GB Dual USB Flash Drive USB M3.0
- SanDisk Ultra 16GB Dual USB Flash Drive USB M3.0

Follow the instructions below before running the Architecture Compliance Suite (ACS) tests.

Build an EFI System Partition

A storage with EFI System Partition (ESP) must exist in the system for the UEFI-SCT related tests to pass.

1. Build an ESP partition for your target

```
kas build meta-arm/kas/corstone1000-${TARGET}.yml:meta-arm/ci/debug.yml --
↪target corstone1000-esp-image
```

2. Locate the `corstone1000-esp-image-corstone1000-${TARGET}.wic` build artefact in `${WORKSPACE}/build/tmp/deploy/images/corstone1000-${TARGET}/`

Use the EFI System Partition

MPS3

1. Connect a USB drive to your development machine.
2. Run the following command on your development machine to discover which device is your USB drive:

   ```
   lsblk
   ```

   The remaining steps assume the USB drive is `/dev/sdb`.

   Warning: Do not mistake your development machine hard drive with the USB drive.

3. Copy the ESP to the USB drive by running the following command:

   ```
   sudo dd \
   if=${WORKSPACE}/build/tmp/deploy/images/corstone1000-mps3/corstone1000-esp-
   ↪image-corstone1000-mps3.wic \
   of=/dev/sdb \
   iflag=direct oflag=direct status=progress bs=512; sync;
   ```

4. Plug the USB drive to the MPS3.

FVP

The ESP disk image will automatically be used by the Corstone-1000 FVP as the 2nd MMC card image. It will be used when the SystemReady-IR tests is performed on the FVP in the later section.

Run SystemReady IR ACS Tests

ACS is used to ensure architectural compliance across different implementations of the architecture. Arm Enterprise ACS includes a set of examples of the invariant behaviors that are provided by a set of specifications for enterprise systems (i.e. SBSA, SBBR, etc.). Implementers can verify if these behaviors have been interpreted correctly.

The following test suites and bootable applications are under the `BOOT` partition of the ACS image:

- SCT
- FWTS
- BSA UEFI
- BSA linux
- GRUB
- UEFI manual capsule application

See the directory structure of the ACS image `BOOT` partition below:

```
├── acs_results
├── EFI
│   └── BOOT
│       ├── app
```

(continues on next page)

(continued from previous page)

```
│       ├── bbr
│       ├── bootaa64.efi
│       ├── bsa
│       ├── debug
│       ├── grub.cfg
│       ├── Shell.efi
│       ├── sie_startup.nsh
│       └── startup.nsh
├── Image
├── security-interface-extension-keys
│   ├── NullPK.auth
│   ├── TestDB1.auth
│   ├── TestDB1.der
│   ├── TestDBX1.auth
│   ├── TestDBX1.der
│   ├── TestKEK1.auth
│   ├── TestKEK1.der
│   ├── TestPK1.auth
│   └── TestPK1.der
├── startup.nsh
└── yocto_image.flag
```

The `BOOT` partition is also used to store test results in the `acs_results` folder.

Important: Ensure that the `acs_results` folder is empty before starting the test.

This sections below describe how to build and run ACS tests on Corstone-1000.

1. On your host development machine, clone the Arm SystemReady ACS repository.

   ```
   cd ${WORKSPACE}
   git clone https://github.com/ARM-software/arm-systemready.git -b v23.09_SR_
   ↪REL2.0.0_ES_REL1.3.0_IR_REL2.1.0 --depth 1
   ```

   This repository contains the infrastructure to build the ACS and the bootable prebuilt images to be used for the certifications of SystemReady IR.

2. Find the pre-built ACS live image in `${WORKSPACE}/arm-systemready/IR/prebuilt_images/v23.09_2.1.0/ir-acs-live-image-generic-arm64.wic.xz`.

   Note: This prebuilt ACS image includes v5.13 kernel, which does not provide USB driver support for Corstone-1000.

3. Decompress the pre-built ACS live image.

   ```
   cd ${WORKSPACE}/arm-systemready/IR/prebuilt_images/v23.09_2.1.0
   unxz ir-acs-live-image-generic-arm64.wic.xz
   ```

MPS3

1. Connect a USB drive (other than the one used for the ESP) to the host development machine.
2. Run the following command to discover which device is your USB drive:

   ```
   lsblk
   ```

   The remaining steps assume the USB drive is `/dev/sdc`.

   Warning: Do not mistake your development machine hard drive with the USB drive.

3. Copy the ACS image to the USB drive by running the following commands:

   ```
   cd ${WORKSPACE}/arm-systemready/IR/prebuilt_images/v23.09_2.1.0
   sudo dd if=ir-acs-live-image-generic-arm64.wic of=/dev/sdc iflag=direct␣
   ↪oflag=direct bs=1M status=progress; sync
   ```

4. Plug the USB drive to the MPS3. At this point you should have both the USB drive with the ESP and the USB drive with the ACS image plugged to the MPS3.
5. Reboot the MPS3.

The MPS3 will reset multiple times during the test, and it might take approximately 24 to 36 hours to finish the test.

Important: Unplug the ESP USB drive from the MPS3 if it is preventing GRUB from finding the bootable partition. Leave only the ACS image USB drive plugged in to run the ACS tests.

The ESP USB drive can be plugged in again after selecting the *Linux Boot* option in the GRUB menu at the end of the ACS tests.

Warning: A timeout issue has been observed while booting Linux during the ACS tests, causing the system to boot into emergency mode. Booting Linux is necessary to run certain tests, such as *dt-validation*. The following workaround is required to enable Linux to boot properly and perform all Linux-based tests:

1. Press Enter at the Linux prompt.
2. Open the file */etc/systemd/system.conf* and set *DefaultDeviceTimeoutSec=infinity*.
3. Reboot the platform using the *reboot* command.
4. Select the *Linux Boot* option from the GRUB menu.
5. Allow Linux to boot and run the remaining ACS tests until completion.

FVP

1. Run `tmux`:

```
cd ${WORKSPACE} && tmux
```

2. Run the commands below within `tmux` to run the ACS test on FVP using the built firmware image and the pre-built ACS image identified above:

```
./meta-arm/scripts/runfvp \
--terminals=tmux \
./build/tmp/deploy/images/corstone1000-fvp/corstone1000-flash-firmware-
↪image-corstone1000-fvp.fvpconf \
-- -C board.msd_mmc.p_mmc_file=${WORKSPACE}/arm-systemready/IR/prebuilt_
↪images/v23.09_2.1.0/ir-acs-live-image-generic-arm64.wic
```

Note: The FVP will reset multiple times during the test. The ACS tests might take up to 1 day to complete when run on FVP.

The message *ACS run is completed* will be displayed on the FVP host terminal when the test runs to completion. You will be prompted to press the Enter key to access the Linux prompt.

Test Sequence and Results

U-Boot should be able to boot the GRUB bootloader from the first partition.

If GRUB is not interrupted, the tests are executed automatically in the following order:

- SCT
- UEFI BSA
- FWTS

The results can be fetched from the *acs_results* folder in the `BOOT` partition of the USB drive (for MPS3) or SD Card (for FVP).

Note: Access the *acs_results* folder in FVP by running the following commands:

```
sudo mkdir /mnt/test
sudo mount -o rw,offset=1048576 \
${WORKSPACE}/arm-systemready/IR/prebuilt_images/v23.09_2.1.0/ir-acs-live-image-generic-
↪arm64.wic \
/mnt/test
```

1.3.4 Capsule Update

Warning: The **Corstone-1000 with Cortex-A320 FVP** becomes unresponsive when the Linux kernel driver for the Ethos-U85 NPU loads automatically after a software reboot. This behavior results from a power reset control issue in the **Corstone-1000 with Cortex-A320 FVP** model. To prevent the failure and complete the test successfully, rebuild the **Corstone-1000 with Cortex-A320** firmware image using the following steps:

1. Clone the *systemready-patch* repository to your `${WORKSPACE}`.

```
cd ${WORKSPACE}
git clone https://git.gitlab.arm.com/arm-reference-solutions/systemready-
↪patch.git \
-b CORSTONE1000-2025.12
```

2. Re-Build the **Corstone-1000 with Cortex-A320 FVP** software stack as follows:

```
kas build meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml:meta-
↪arm/kas/corstone1000-a320.yml:\
systemready-patch/embedded-a/corstone1000/disable_module_autoloading/
↪disable_module_autoloading.yml
```

Important: Payload GUIDs (`${BL2_GUID}`, `${TFM_S_GUID}`, `${FIP_GUID}`, and `${INITRAMFS_GUID}`) are different depending on whether the capsule is built for the `fvp` or `mps3` `${TARGET}`.

Payloads	FVP	MPS3
BL2	f1d883f9-dfeb-5363-98d8-686ee3b69f4f	fbfbefaa-0a56-50d5-b651-74091d3d62cf
TFM_S	7fad470e-5ec5-5c03-a2c1-4756b495de61	af4cc7ad-ee2e-5a39-aad5-fac8a1e6173c
FIP	f1933675-5a8c-5b6d-9ef4-846739e89bc8	55302f96-c4f0-5cf9-8624-e7cc388f2b68
INITRAMFS	f771aff9-c7e9-5f99-9eda-2369dd694f61	3e8ac972-c33c-5cc9-90a0-cdd3159683ea

The following section describes the steps to update the firmware using Capsule Update as the Corstone-1000 supports UEFI.

The firmware update process is tested with an invalid capsule and with valid capsules to validate the robustness and error-handling capabilities of the firmware update mechanism.

Positive full capsule update test: The Corstone-1000 is provided with a valid full capsule, which it applies successfully. The system then boots normally and reaches the Linux command prompt.

Positive partial capsule update test: The Corstone-1000 is provided with a valid partial capsule that specifies an update for a single component only. The capsule is applied successfully, after which the system boots normally and reaches the Linux command prompt.

Rollback protection capsule update test: The Corstone-1000 is provided with an outdated capsule containing lower version numbers for all payloads. The capsule is correctly rejected due to rollback protection, and the previously installed firmware is retained.

Three different capsules are therefore needed to perform the tests.

The following payloads can be individually updated:

- Boot Loader stage 2 (BL2)
- Trusted Firmware-M Secure partition (TFM_S)

- Firmware Image Package (FIP)
- Initial RAM Filesystem (INITRAMFS)

Generate Capsules

EDK II's `GenerateCapsule` tool is used to generate capsules and is built automatically for the host machine during the firmware image building process. The tool can be found at `${WORKSPACE}/build/tmp/sysroots-components/aarch64/edk2-basetools-native/usr/bin/edk2-BaseTools/BinWrappers/PosixLike/GenerateCapsule`.

A JSON file containing metadata about the capsule payloads needs to be created using the script found at `${WORKSPACE}/meta-arm/meta-arm/scripts/generate_capsule_json_multiple.py`. This JSON file is required by EDK II's `GenerateCapsule` tool to generate the capsule.

The capsule's default metadata passed can be found in the `${WORKSPACE}/meta-arm/meta-arm-bsp/recipes-bsp/images/corstone1000-flash-firmware-image.bb` and `${WORKSPACE}/meta-arm/kas/corstone1000-image-configuration.yml` files.

Valid Full Capsule

An automatically generated capsule can be found at `${WORKSPACE}/build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000-${TARGET}-v6.uefi.capsule` after running a firmware build.

The default metadata values are assumed to be correct to generate a valid capsule.

This capsule will be used for the positive capsule update test.

Valid Partial Capsule

To generate a capsule that updates only a single component, explicitly set the firmware version for that component and mark it as the only payload to be updated.

The **partial capsule** is also valid, but sets the firmware version to **7 only** for the **BL2** component, indicating that no other components should be updated.

Use the following commands to generate the *capsule_config.json* file, which is required by the EDK2 tool for capsule creation:

```
cd ${WORKSPACE}

python3 meta-arm/meta-arm/scripts/generate_capsule_json_multiple.py \
--selected_components DUMMY_START  BL2  DUMMY_END \
--components DUMMY_START  BL2  TFM_S  FIP  INITRAMFS  DUMMY_END \
--fw_versions 0 7 0 0 0 0 \
--guids \
6f784cbf-7938-5c23-8d6e-24d2f1410fa9  \
${BL2_GUID} ${TFM_S_GUID}  ${FIP_GUID}  ${INITRAMFS_GUID} \
b57e432b-a250-5c73-93e3-90205e64baba \
--hardware_instances 1  1  1  1  1  1 \
--lowest_supported_versions 5 5 5 5 5 5 \
--monotonic_counts 1  1  1  1  1  1 \
--payloads \
build/tmp/work/corstone1000_${TARGET}-poky-linux-musl/corstone1000-flash-firmware-image/
```

(continues on next page)

(continued from previous page)

```
↪1.0/sources/corstone1000-flash-firmware-image-1.0/dummy.bin \
build/tmp/deploy/images/corstone1000-${TARGET}/bl2_signed.bin \
build/tmp/deploy/images/corstone1000-${TARGET}/tfm_s_signed.bin \
build/tmp/deploy/images/corstone1000-${TARGET}/signed_fip-corstone1000.bin \
build/tmp/deploy/images/corstone1000-${TARGET}/Image.gz-initramfs-corstone1000-${TARGET}.
↪bin \
build/tmp/work/corstone1000_${TARGET}-poky-linux-musl/corstone1000-flash-firmware-image/
↪1.0/sources/corstone1000-flash-firmware-image-1.0/dummy.bin \
--update_image_indexes 5  1  2  3  4  6 \
--private_keys \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
--certificates \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
--output capsule_config.json
```

Run the command below to generate the partial capsule:

```
./build/tmp/sysroots-components/aarch64/edk2-basetools-native/usr/bin/edk2-BaseTools/
↪BinWrappers/PosixLike/GenerateCapsule \
-e \
-j capsule_config.json \
--capflag PersistAcrossReset \
-o corstone1000-${TARGET}-partial-v7.uefi.capsule
```

The partial capsule will be located in the `${WORKSPACE}` directory.

Invalid Capsule

Generate a capsule with firmware version metadata for all payloads set lower than that of a valid capsule. The valid capsule has a default firmware version of 6 for all payloads, while the simulated invalid capsule has the firmware version set to 5 for all payloads.

Use the following commands to generate the *capsule_config.json* file, which is required by the EDK2 tool for capsule creation:

```
cd ${WORKSPACE}

python3 meta-arm/meta-arm/scripts/generate_capsule_json_multiple.py \
--selected_components DUMMY_START  BL2  TFM_S  FIP  INITRAMFS  DUMMY_END \
--components DUMMY_START  BL2  TFM_S  FIP  INITRAMFS  DUMMY_END \
--fw_versions 5 5 5 5 5 5 \
--guids \
```

(continues on next page)

(continued from previous page)

```
6f784cbf-7938-5c23-8d6e-24d2f1410fa9  \
${BL2_GUID} ${TFM_S_GUID}  ${FIP_GUID}  ${INITRAMFS_GUID} \
b57e432b-a250-5c73-93e3-90205e64baba \
--hardware_instances 1  1  1  1  1  1 \
--lowest_supported_versions 5 5 5 5 5 5 \
--monotonic_counts 1  1  1  1  1  1 \
--payloads \
build/tmp/work/corstone1000_${TARGET}-poky-linux-musl/corstone1000-flash-firmware-image/
↪1.0/sources/corstone1000-flash-firmware-image-1.0/dummy.bin \
build/tmp/deploy/images/corstone1000-${TARGET}/bl2_signed.bin \
build/tmp/deploy/images/corstone1000-${TARGET}/tfm_s_signed.bin \
build/tmp/deploy/images/corstone1000-${TARGET}/signed_fip-corstone1000.bin \
build/tmp/deploy/images/corstone1000-${TARGET}/Image.gz-initramfs-corstone1000-${TARGET}.
↪bin \
build/tmp/work/corstone1000_${TARGET}-poky-linux-musl/corstone1000-flash-firmware-image/
↪1.0/sources/corstone1000-flash-firmware-image-1.0/dummy.bin \
--update_image_indexes 5  1  2  3  4  6 \
--private_keys \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_key.key \
--certificates \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000_capsule_cert.crt \
--output capsule_config.json
```

Run the command below to generate the invalid capsule:

```
./build/tmp/sysroots-components/aarch64/edk2-basetools-native/usr/bin/edk2-BaseTools/
↪BinWrappers/PosixLike/GenerateCapsule \
-e \
-j capsule_config.json \
--capflag PersistAcrossReset \
-o corstone1000-${TARGET}-v5.uefi.capsule
```

The invalid capsule will be located in the `${WORKSPACE}` directory.

Transfer Capsules to Target

The capsule delivery process described below is the direct method (usage of capsules from the ACS image) as opposed to the on-disk method (delivery of capsules using a file on a mass storage device).

MPS3

1. Prepare a USB drive as explained in *this* section.
2. Copy the capsule file to the root directory of the `BOOT` partition in the USB drive.

```
cp ${WORKSPACE}/build/tmp/deploy/images/corstone1000-mps3/corstone1000-mps3-v6.
↪uefi.capsule /dev/sdc/BOOT/
cp ${WORKSPACE}/corstone1000-mps3-v5.uefi.capsule /dev/sdc/EFI/BOOT/
cp ${WORKSPACE}/corstone1000-mps3-partial-v7.uefi.capsule /dev/sdc/EFI/BOOT/
sync
```

Note: `/dev/sdc` is the assumed path for the ACS Image USB drive. Replace it with the actual device path as enumerated on your development machine.

Important: The direct Capsule Update method requires that the capsule files not be placed in the `EFI/UpdateCapsule` directory, as doing so might inadvertently trigger the on-disk update method.

FVP

1. Download and extract the ACS image *as described for the MPS3*. The ACS image extraction location will be referred below as `${ACS_IMAGE_PATH}`.

   Note: Creating a USB drive with the ACS image is not required as the image will be mounted with the steps below.

2. Find the first partition's offset of the `ir-acs-live-image-generic-arm64.wic` image using the `fdisk` tool. The partition table can be listed using:

```
fdisk -lu ${ACS_IMAGE_PATH}/ir-acs-live-image-generic-arm64.wic
Device                                                Start     End␣
↪Sectors  Size Type
${ACS_IMAGE_PATH}/ir-acs-live-image-generic-arm64.wic1    2048  309247 ␣
↪307200  150M Microsoft basic data
${ACS_IMAGE_PATH}/ir-acs-live-image-generic-arm64.wic2  309248 1343339␣
↪1034092  505M Linux filesystem
```

   Given that the first partition starts at sector 2048 and each sector is 512 bytes in size, the first partition is at offset 1048576 (2048 x 512).

3. Mount the `ir-acs-live-image-generic-arm64.wic` image using the previously calculated offset:

```
sudo mkdir /mnt/ir-acs-live-image-generic-arm64
sudo mount -o rw,offset=<first_partition_offset> ${ACS_IMAGE_PATH}/ir-acs-
↪live-image-generic-arm64.wic  /mnt/ir-acs-live-image-generic-arm64
```

4. Copy the capsules:

```
sudo cp ${WORKSPACE}/build/tmp/deploy/images/corstone1000-fvp/corstone1000-
↪fvp-v6.uefi.capsule /mnt/ir-acs-live-image-generic-arm64/
sudo cp ${WORKSPACE}/corstone1000-fvp-v5.uefi.capsule /mnt/ir-acs-live-
↪image-generic-arm64/
sudo cp ${WORKSPACE}/corstone1000-fvp-partial-v7.uefi.capsule /mnt/ir-acs-
↪live-image-generic-arm64/
sync
```

5. Unmount the IR image:

```
sudo umount /mnt/ir-acs-live-image-generic-arm64
```

Run Capsule Update Tests

The valid capsules will be used first to run the positive capsule update tests. This will be followed by using the invalid capsule to run the rollback protection capsule update test.

Important: This sequence order must be respected as the invalid capsule has a firmware version lower than the firmware version in the valid capsule. The rollback protection capsule update test effectively tests that firmware rollback is not permitted.

Positive Full Capsule Update Test

1. Run Corstone-1000 with the ACS image containing the two capsule files:
   - MPS3:
     1. Plug the prepared USB drive which has the IR prebuilt image and two capsules to the MPS3.
     2. Power cycle the MPS3.
   - FVP:
     1. Run `tmux`:

```
cd ${WORKSPACE} && tmux
```

     1. Run the FVP within `tmux` with the IR prebuilt image which now also contains the two capsules:

```
kas shell meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml \
-c "../meta-arm/scripts/runfvp --terminals=tmux \
-- -C board.msd_mmc.p_mmc_file=${ACS_IMAGE_PATH}/ir-acs-live-image-generic-
↪arm64.wic"
```

> **Warning:** `${ACS_IMAGE_PATH}` must be an absolute path. Ensure there are no spaces before or after of = of the `-C board.msd_mmc.p_mmc_file` option.

2. Wait until U-Boot loads EFI from the ACS image and interrupt the EFI shell by pressing the `Escape` key when the following prompt is displayed on the Host Processor terminal (`ttyUSB2` for MPS3).

```
Press ESC in 4 seconds to skip startup.nsh or any other key to continue.
```

3. The content of the first file system (`File System 0`), where the capsule files were copied, can be accessed by running the following command:

```
FS0:
```

4. Run the `CapsuleApp` application with the valid capsule file:

   - MPS3:

```
EFI/BOOT/app/CapsuleApp.efi EFI/BOOT/corstone1000-mps3-v6.uefi.
↪capsule
```

   - FVP:

```
EFI/BOOT/app/CapsuleApp.efi corstone1000-fvp-v6.uefi.capsule
```

   The capsule update will be started.

Note: The capsule update takes about 8 minutes to complete on MPS3 and between 15-30 minutes on FVP.

The Corstone-1000 will reset after successfully applying the capsule.

The software stack copies the capsule content to the external flash, which is shared between the Secure Enclave and the Host Processor before rebooting the system.

After the first reboot, TrustedFirmware-M should apply the valid capsule and display the following log on the Secure Enclave terminal (`ttyUSB1` for MPS3) before rebooting the system a second time:

```
...
SysTick_Handler: counted = 10, expiring on = 360
SysTick_Handler: counted = 20, expiring on = 360
SysTick_Handler: counted = 30, expiring on = 360
...
metadata_write: success: active = 1, previous = 0
flash_full_capsule: exit
corstone1000_fwu_flash_image: exit: ret = 0
...
```

The above log snippet indicates that the new capsule image is successfully applied, and the board is booting with the external flash's Bank-1.

After a second reboot, the following log should be displayed on on the Secure Enclave terminal (`ttyUSB1`):

```
...
fmp_set_image_info:133 Enter
FMP image update: image id = 0
FMP image update: status = 0version=6 last_attempt_version=6.
fmp_set_image_info:157 Exit.
corstone1000_fwu_host_ack: exit: ret = 0
...
```

5. Interrupt the U-Boot shell.

```
Hit any key to stop autoboot:
```

6. Run the following commands in order to run the Corstone-1000 Linux kernel.

Note: Otherwise, the execution ends up in the ACS live image.

```
$ unzip $kernel_addr 0x90000000
$ loadm 0x90000000 $kernel_addr_r $filesize
$ bootefi $kernel_addr_r $fdtcontroladdr
```

7. The first boot after a capsule update is considered the trial stage, during which the FWU image is accepted. However, to view the updated contents of the EFI System Resource Table (ESRT), an additional reboot is required.

```
# reboot
```

8. Interrupt the U-Boot shell when prompted.

```
Hit any key to stop autoboot:
```

9. Run the following commands to boot the Corstone-1000 Linux kernel.

Note: If these commands are not executed, the system will default to booting into the ACS live image.

```
$ unzip $kernel_addr 0x90000000
$ loadm 0x90000000 $kernel_addr_r $filesize
$ bootefi $kernel_addr_r $fdtcontroladdr
```

10. Once the system has fully booted again, *read the ESRT* to confirm that the firmware version reflects the updated capsule.

Warning: Do not terminate FVP between the positive full capsule update and partial capsule update tests.

Positive Partial Capsule Update Test

Follow the steps for the *positive full capsule update test* ensuring you use `corstone1000-${TARGET}-partial-v7.uefi.capsule` instead of `corstone1000-${TARGET}-v6.uefi.capsule`.

Once the system has fully booted again, *read the ESRT* to confirm that the firmware version reflects the updated capsule.

Warning: Do not terminate FVP between the positive partial capsule update rollback protection capsule update tests.

Rollback Protection Capsule Update Test

Important: The *positive partial capsule update test* must be run before running the rollback protection capsule update test.

1. After running the positive capsule update test, reboot the system by typing the following command on the Host Processor terminal (`ttyUSB2` for MPS3):

   ```
   reboot
   ```

2. Wait until U-Boot loads EFI from the ACS image and interrupt the EFI shell by pressing the `Escape` key when the following prompt is displayed on the Host Processor terminal (`ttyUSB2` for MPS3).

   ```
   Press ESC in 4 seconds to skip startup.nsh or any other key to continue.
   ```

3. The content of the first file system (`File System 0`), where the capsule files were copied, can be accessed by running the following command:

   ```
   FS0:
   ```

4. Run the `CapsuleApp` application with the invalid capsule file:

   - MPS3:

     ```
     EFI/BOOT/app/CapsuleApp.efi EFI/BOOT/corstone1000-mps3-v5.uefi.capsule
     ```

   - FVP:

     ```
     EFI/BOOT/app/CapsuleApp.efi corstone1000-fvp-v5.uefi.capsule
     ```

5. TrustedFirmware-M should reject the capsule due to having a lower firmware version and display the following log on the Secure Enclave terminal (`ttyUSB1` for MPS3):

   ```
   ...
     uefi_capsule_retrieve_images: image 0 at 0xa0000070, size=15654928
     uefi_capsule_retrieve_images: exit
     flash_full_capsule: enter: image = 0x0xa0000070, size = 7764541, version␣
   ```

(continues on next page)

(continued from previous page)

```
↪= 5
  ERROR: flash_full_capsule: version error
  private_metadata_write: enter: boot_index = 1
  private_metadata_write: success
  fmp_set_image_info:133 Enter
  FMP image update: image id = 0
  FMP image update: status = 1version=6 last_attempt_version=5.
  fmp_set_image_info:157 Exit.
  corstone1000_fwu_flash_image: exit: ret = -1
  fmp_get_image_info:232 Enter
  pack_image_info:207 ImageInfo size = 105, ImageName size = 34,␣
↪ImageVersionName
  size = 36
  fmp_get_image_info:236 Exit
...
```

The Secure Enclave tries to load the new image a predetermined number of times if the capsule passes initial verification but fails verifications performed during boot time.

```
...
metadata_write: success: active = 0, previous = 1
fwu_select_previous: in regular state by choosing previous active␣
↪bank
...
```

The Secure Enclave eventually reverts back to the previously running image.

6. Reboot manually:

```
Shell> reset
```

7. Interrupt the U-Boot shell.

```
Hit any key to stop autoboot:
```

8. Run the following commands in order to run the Corstone-1000 Linux kernel.

Note: Otherwise, the execution ends up in the ACS live image.

```
$ unzip $kernel_addr 0x90000000
$ loadm 0x90000000 $kernel_addr_r $filesize
$ bootefi $kernel_addr_r $fdtcontroladdr
```

9. The first boot after a capsule update is considered the trial stage, during which the FWU image is rejected. However, to view the updated contents of the ESRT, an additional reboot is required.

```
# reboot
```

10. Interrupt the U-Boot shell when prompted.

```
Hit any key to stop autoboot:
```

11. Run the following commands to boot the Corstone-1000 Linux kernel.

Note: If these commands are not executed, the system will default to booting into the ACS live image.

```
$ unzip $kernel_addr 0x90000000
$ loadm 0x90000000 $kernel_addr_r $filesize
$ bootefi $kernel_addr_r $fdtcontroladdr
```

12. Once the system has fully booted again, *read the ESRT* to confirm that the firmware version reflects the updated capsule.

Verifying Firmware Versions via ESRT

After the system has fully booted, verify that the firmware versions of all applied capsule payloads match those currently installed on the system. This can be done by inspecting the ESRT, which is exposed by the Linux kernel.

Reading ESRT Entries

To read each ESRT entry, use the following commands:

```
cat /sys/firmware/efi/esrt/entries/entry0/*
cat /sys/firmware/efi/esrt/entries/entry1/*
cat /sys/firmware/efi/esrt/entries/entry2/*
cat /sys/firmware/efi/esrt/entries/entry3/*
```

These entries typically correspond to:

- `entry0`: BL2
- `entry1`: TFM_S
- `entry2`: FIP
- `entry3`: INITRAMFS

Note: Entry indices may vary depending on how your firmware capsules are structured. Adjust as needed.

Structure of Each ESRT Entry

Each directory under `/sys/firmware/efi/esrt/entries/entryX/` contains files representing the following fields:

Field Name	Description
`capsule_flags`	Attributes of the update capsule (e.g., persist, reset)
`fw_class`	GUID identifying the firmware component
`fw_type`	Firmware type (e.g., system, device, peripheral)
`fw_version`	Currently installed firmware version
`last_attempt_status`	Status of the last update attempt (e.g., success, failure)
`last_attempt_version`	Version that was last attempted to install
`lowest_supported_fw_version`	Minimum firmware version that is still supported

Verifying an ESRT Entry

To check the version and status of BL2 (`entry0`), run:

```
cat /sys/firmware/efi/esrt/entries/entry0/fw_version
cat /sys/firmware/efi/esrt/entries/entry0/last_attempt_version
cat /sys/firmware/efi/esrt/entries/entry0/last_attempt_status
```

Positive Full Capsule Update Test ESRT

The following table shows the details of the first four ESRT entries for the positive capsule update test:

capsule_fl	fw_class	fw_typ	fw_versi	last_attempt_s	last_attempt_ve	lowest_supported_fw_ver
0	${BL2_GUID}	0	6	0	6	0
0	${TFM_S_GUID}	0	6	0	6	0
0	${FIP_GUID}	0	6	0	6	0
0	${INITRAMFS_(	0	6	0	6	0

Positive Partial Capsule Update Test ESRT

The following table shows the details of the first four ESRT entries for the positive capsule update test:

capsule_fl	fw_class	fw_typ	fw_versi	last_attempt_s	last_attempt_ve	lowest_supported_fw_ver
0	${BL2_GUID}	0	7	0	7	0
0	${TFM_S_GUID}	0	6	0	6	0
0	${FIP_GUID}	0	6	0	6	0
0	${INITRAMFS_(	0	6	0	6	0

Rollback Protection Capsule Update Test ESRT

The following table shows the details of the first four ESRT entries for the rollback protection capsule update test:

capsule_fl	fw_class	fw_typ	fw_versi	last_attempt_s	last_attempt_ve	lowest_supported_fw_ver
0	${BL2_GUID}	0	7	1	5	0
0	${TFM_S_GUID}	0	6	0	6	0
0	${FIP_GUID}	0	6	0	6	0
0	${INITRAMFS_(	0	6	0	6	0

See the UEFI documentation for more information on the significance of the table fields.

1.3.5 Linux Distributions

This sections describes the steps to install major Linux distributions to the Corstone-1000 Host Processor.

The Linux distributions to be installed are:

- Debian
- openSUSE

Follow the instructions below to install the Linux distributions to the Corstone-1000 software stack.

Prepare Installation Media

The media containing the bootable files required to start the installation process needs to be prepared.

Follow the instructions below to create the installation media.

1. Using your development machine, download one of following Linux distribution images:
    - Debian installer image
    - openSUSE Leap installer image

    Note: The location of the ISO file on the development machine will be referred to as `${DISTRO_INSTALLER_ISO_PATH}`.

    ---

2. Create the installation media which will contain the necessary files to install the operation system.

- **MPS3**:
    1. Plug a blank USB drive formatted with FAT32, ensuring it has a minimum capacity of 4GB, to the development machine.
    2. Run the following command to discover which device is your USB drive:

        ```
        lsblk
        ```

        The remaining steps assume the USB drive is `/dev/sdb`.

        Warning: Do not mistake your development machine hard drive with the USB drive.

    3. Write one of the distribution installer ISO file to the USB drive.

        ```
        sudo dd if=${DISTRO_INSTALLER_ISO_PATH} of=/dev/sdb iflag=direct␣
        ↪oflag=direct status=progress bs=1M; sync;
        ```

- **FVP**:

    The distribution installer ISO file does not need to be burnt to a USB drive. It will be used as is when starting the FVP install the distribution.

Prepare System Drive

A system (or boot) drive, to store all the operating system files and used to boot the distribution, is required as Corstone-1000 on-board non-volatile storage size is insufficient for installing the distributions.

- **MPS3:**
    1. Find another blank USB drive formatted with FAT32 with a minimum capacity of 4GB.
    2. Do not yet connect this blank USB drive to the MPS3. It will be used as the primary drive to boot the distribution.
- **FVP:**
    1. Create an 10 GB GUID Partition Table (GPT) formatted MultiMediaCard (MMC) image.

        ```
        dd if=/dev/zero of=${WORKSPACE}/fvp_distro_system_drive.img \
        bs=1 count=0 seek=10G; sync; \
        parted -s fvp_distro_system_drive.img mklabel gpt
        ```

    2. This MMC image will be used as the primary drive to boot the distribution.

Installation

MPS3

1. Connect the installation media, which contains the installer for the desired distribution, to the MPS3.
2. Open a serial port terminal interface to `/dev/ttyUSB0` in one terminal window on your development machine.

    ```
    sudo picocom -b 115200 /dev/ttyUSB0
    ```

3. Open a serial port terminal interface to `/dev/ttyUSB2` in another terminal window on your development machine.

    ```
    sudo picocom -b 115200 /dev/ttyUSB2
    ```

4. When the installation screen is displayed on `ttyUSB2`, plug in the (still empty) system drive to the MPS3.
5. Start the distribution installation process.

> **Note:** Reboot the MPS3 with both USB drives (installation media and empty system drive) connected to it if the distribution installer does not start.

Note: Due to the performance limitation, the distribution installation process can take up to 24 hours to complete.

FVP

1. Run the `tmux`:

   ```
   cd ${WORKSPACE} && tmux
   ```

2. Start the FVP within `tmux` with the system drive as the primary drive and the distro ISO file as the secondary drive:

   ```
   kas shell meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml \
   -c "../meta-arm/scripts/runfvp --terminals=tmux -- \
   -C board.msd_mmc.p_mmc_file=${WORKSPACE}/fvp_distro_system_drive.img \
   -C board.msd_mmc_2.p_mmc_file=${DISTRO_INSTALLER_ISO_PATH}"
   ```

   The Linux distribution will be installed on `fvp_distro_system_drive.img`.

Debian Installation Extra Steps

Debian installation may need some extra steps, that are indicated below:

1. Answer `Yes` to the question `Force grub installation to the EFI removable media path?`.

   If the GRUB installation fails, these are the steps to follow on the subsequent popups:

   1. Select `Continue`, then `Continue` again on the next popup.
   2. Scroll down and select `Execute a shell`.
   3. Select `Continue`.
   4. Enter the following command:

      ```
      in-target grub-install --no-nvram --force-extra-removable
      ```

   5. Enter the following command:

      ```
      in-target update-grub
      ```

   6. Enter the following command:

      ```
      exit
      ```

   7. Select `Continue without boot loader`, then select `Continue` on the next popup.
   8. At this stage, the installation should proceed as normal.

2. Answer `No` to the question `Update NVRAM variables to automatically boot into Debian?`.

Boot Distribution

- MPS3
  1. Once the installation is complete, unplug the installation media.
  2. Perform a cold boot of the MPS3.
- FVP

  The target should automatically boot into the installed operating system image.

  Stop the FVP with `CTRL+C` and run `tmux`:

  ```
  cd ${WORKSPACE} && tmux
  ```

  Run the command below to simulate a cold boot:

  ```
  kas shell meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml \
  -c "../meta-arm/scripts/runfvp --terminals=tmux -- \
  -C board.msd_mmc.p_mmc_file=${WORKSPACE}/fvp_distro_system_drive.img"
  ```

  Warning: To manually enter recovery mode, once the FVP begins booting, you can quickly change the boot option in GRUB, to boot into recovery mode. This option will disappear quickly, so it is best to preempt it.

  Select `Advanced Options for <OS>` and then `<OS> (recovery mode)`.

The target will then enter recovery mode, from which the user can access a shell after entering the password for the `root` user.

Timeout Optimizations

Important: Operating system timeouts are inconsistent across systems. Skip this section if the system boots to Debian or openSUSE without any issue.

Make the system modification below whilst in recovery mode to increase timeouts and boot to the installed distribution.

1. Remove the timeout limit for device operations.
   - **Debian**

     ```
     vi /etc/systemd/system.conf
     DefaultDeviceTimeoutSec=infinity
     ```

   - **openSUSE**

     ```
     vi /usr/lib/systemd/system.conf
     DefaultDeviceTimeoutSec=infinity
     ```

     Warning: As modifying `system.conf` in `/usr/lib/systemd/` is not working as it is getting overwritten, copy `system.conf` from `/usr/lib/systemd/` to `/etc/systemd/system.conf.d/` after the above edit.

2. Set the maximum time that the system will wait for a user to successfully log in before timing out to 180 seconds.

   - **Debian**

     ```
     vi /etc/login.defs
     LOGIN_TIMEOUT   180
     ```

   - **openSUSE**

     ```
     vi /usr/etc/login.defs
     LOGIN_TIMEOUT   180
     ```

3. Ensure the changes are applied by run the command below.

   ```
   systemctl daemon-reload
   ```

4. Perform a cold boot of the target.

Log into the Distribution

Login with the `root` username and its corresponding password (set during installation) at the distribution login prompt after booting. See an illustration for Debian below:

```
debian login:
```

1.3.6 UEFI Secure Boot

The UEFI Secure Boot test is designed to verify the integrity and authenticity of the system's boot process. This test ensures that only trusted, signed images are executed, thereby preventing unauthorized or malicious code from running. A successful test confirms that the signed image executes correctly, while any unsigned image is blocked from running.

Generate Keys, Signed Image and Unsigned Image

1. Build an EFI System Partition as described *here*.
2. Clone the *systemready-patch* repository to your workspace.

   ```
   cd ${WORKSPACE}

   git clone https://gitlab.arm.com/arm-reference-solutions/systemready-patch \

   -b CORSTONE1000-2025.12
   ```

3. Set the current working directory to build directory's subdirectory containing the software stack build images.

   ```
   cd ${WORKSPACE}/build/tmp/deploy/images/corstone1000-${TARGET}/
   ```

4. Run the image signing script (without changing the current working directory).

   ```
   ./${WORKSPACE}/systemready-patch/embedded-a/corstone1000/secureboot/create_
   ↪keys_and_sign.sh \
   -d ${TARGET} \
   -v ${CERTIFICATE_VALIDITY_DURATION_IN_DAYS}
   ```

Important: The efitools package is required to execute the script.

`${CERTIFICATE_VALIDITY_DURATION_IN_DAYS}` is an integer that specifies the certificate's validity period in days.

Note: Consult the image signing script help message (`-h`) for more information about other optional arguments.

The script is interactive and contains commands that require `sudo` level permissions.

The keys, signed kernel image, and unsigned kernel image will be copied to the exisiting ESP image. The modified ESP image can be found at `${WORKSPACE}/build/tmp/deploy/images/corstone1000-${TARGET}/corstone1000-esp-image-corstone1000-${TARGET}.wic`.

Run Unsigned Image Boot Test

MPS3

1. Follow the instructions *here* to use the ESP.
2. Perform a cold boot of the MPS3.
3. On the Host Processor terminal host side, stop the execution of U-Boot when prompted to do so with the message `Press any key to stop`.

   Warning: There is a timeout of 3 seconds to stop the execution at the U-Boot prompt.

   The U-Boot console prompt looks as follows:

   ```
   corstone1000#
   ```

   Important: The rest of the instructions below will be executed on the U-Boot terminal.

4. On the U-Boot console, reset USB.

   ```
   corstone1000# usb reset
   resetting USB...
   Bus usb@40200000: isp1763 bus width: 16, oc: not available
   USB ISP 1763 HW rev. 32 started
   scanning bus usb@40200000 for devices... port 1 high speed
   3 USB Device(s) found
          scanning usb for storage devices... 1 Storage Device(s) found
   ```

   Note: Occasionally, the USB reset may fail to detect the USB device. It is advisable to rerun the USB reset command.

5. Select the first USB device, which should be the USB drive containing the ESP.

```
corstone1000# usb dev 0
```

6. Enroll the four UEFI secure boot authenticated variables.

```
corstone1000# \
load usb 0 $loadaddr corstone1000_secureboot_keys/PK.auth && setenv -e -nv -
↪bs -rt -at -i $loadaddr:$filesize PK; \
load usb 0 $loadaddr corstone1000_secureboot_keys/KEK.auth && setenv -e -nv␣
↪-bs -rt -at -i $loadaddr:$filesize KEK; \
load usb 0 $loadaddr corstone1000_secureboot_keys/db.auth && setenv -e -nv -
↪bs -rt -at -i $loadaddr:$filesize db; \
load usb 0 $loadaddr corstone1000_secureboot_keys/dbx.auth && setenv -e -nv␣
↪-bs -rt -at -i $loadaddr:$filesize dbx
```

7. Attempt to Load the unsigned kernel image.

```
corstone1000# \
load usb 0 $loadaddr corstone1000_secureboot_mps3_images/Image_mps3
loadm $loadaddr $kernel_addr_r $filesize
bootefi $kernel_addr_r $fdtcontroladdr

Booting /MemoryMapped(0x0,0x88200000,0x236aa00)
Image not authenticated
Loading image failed
```

The unsigned Linux kernel image should not be loaded.

FVP

1. Follow the instructions *here* to use the ESP.
2. Run the software stack as described *here*.
3. On the Host Processor terminal host side, stop the execution of U-Boot when prompted to do so with the message `Press any key to stop`.

> **Warning:** There is a timeout of 3 seconds to stop the execution at the U-Boot prompt.

The U-Boot console prompt looks as follows:

```
corstone1000#
```

> **Important:** The rest of the instructions below will be executed on the U-Boot terminal.

4. On the U-Boot console, set the current MMC device.

```
corstone1000# mmc dev 1
```

5. Enroll the four UEFI secure boot authenticated variables.

```
corstone1000# \
load mmc 1:1 $loadaddr corstone1000_secureboot_keys/PK.auth && setenv -e -
↪nv -bs -rt -at -i $loadaddr:$filesize PK; \
load mmc 1:1 $loadaddr corstone1000_secureboot_keys/KEK.auth && setenv -e -
↪nv -bs -rt -at -i $loadaddr:$filesize KEK; \
load mmc 1:1 $loadaddr corstone1000_secureboot_keys/db.auth && setenv -e -
↪nv -bs -rt -at -i $loadaddr:$filesize db; \
load mmc 1:1 $loadaddr corstone1000_secureboot_keys/dbx.auth && setenv -e -
↪nv -bs -rt -at -i $loadaddr:$filesize dbx
```

6. Attempt to Load the unsigned kernel image.

```
corstone1000# \
load mmc 1:1 $loadaddr corstone1000_secureboot_fvp_images/Image_fvp; \
loadm $loadaddr $kernel_addr_r $filesize; \
bootefi $kernel_addr_r $fdtcontroladdr

Booting /MemoryMapped(0x0,0x88200000,0x236aa00)
Image not authenticated
Loading image failed
```

The unsigned Linux kernel image should not be loaded.

Run Signed Image Boot Test

MPS3

Important: You must first perform the *Unsigned Image Boot Test*.

Load the signed kernel image.

```
corstone1000# \
load usb 0 $loadaddr corstone1000_secureboot_mps3_images/Image_mps3.signed; \
loadm $loadaddr $kernel_addr_r $filesize; \
bootefi $kernel_addr_r $fdtcontroladdr
```

The signed Linux kernel image should be booted successfully.

FVP

Important: You must first perform the *Unsigned Image Boot Test*.

Load the signed kernel image.

```
corstone1000# \
load mmc 1:1 $loadaddr corstone1000_secureboot_fvp_images/Image_fvp.signed; \
loadm $loadaddr $kernel_addr_r $filesize; \
bootefi $kernel_addr_r $fdtcontroladdr
```

The signed Linux kernel image should be booted successfully.

Disable Secure Boot

Running the UEFI Secure Boot Test steps stores UEFI authenticated variables in the secure flash. As a result, U-Boot reads these variables and verifies the Linux kernel image before executing it at each reboot.

In a typical boot scenario, the Linux kernel image is not signed, which will prevent the system from booting due to failed image authentication. To resolve this, the Platform Key (one of the UEFI authenticated variables for secure boot) needs to be deleted.

1. Perform a cold boot of the MPS3.
2. On the Host Processor terminal host side, stop the execution of U-Boot when prompted to do so with the message `Press any key to stop`.
3. On the U-Boot console, delete the Platform Key (PK).
   - MPS3

     ```
     corstone1000# \
     usb reset; \
     usb dev 0; \
     load usb 0 $loadaddr corstone1000_secureboot_keys/PK_delete.auth &&␣
     ↪setenv -e -nv -bs -rt -at -i $loadaddr:$filesize PK; \
     boot
     ```

   - FVP

     ```
     corstone1000# \
     mmc dev 1; \
     load mmc 1:1 $loadaddr corstone1000_secureboot_keys/PK_delete.auth &&␣
     ↪setenv -e -nv -bs -rt -at -i $loadaddr:$filesize PK; \
     boot
     ```

1.3.7 PSA API

The following tests the implementation of the Application Programming Interface (API) of the Platform Security Architecture (PSA) certification scheme. It uses Arm Firmware Framework for Arm A-profile (FF-A) to communicate between the normal world and the secure world to run the Arm Platform Security Architecture Test Suite.

The tests use the *arm_tstee* driver to access Trusted Services Secure Partitions from user space. The driver is included in the Linux Kernel, starting from v6.10.

Important: Ensure there are no USB drives connected to the board when running the test on the MPS3.

The steps below are applicable to both MPS3 and FVP.

1. Start the Corstone-1000 and wait until it boots to Linux on the Host Processor terminal (`ttyUSB2`).
2. Run the PSA API tests by running the commands below in the order shown:

   ```
   psa-iat-api-test
   psa-crypto-api-test
   ```

(continues on next page)

(continued from previous page)

```
psa-its-api-test
psa-ps-api-test
```

1.3.8 External System Processor

Important: Access to the External System Processor is disabled by default. Ensure you are running a software stack image with access to the External System Processor enabled following the steps *here*.

The Linux operating system running on the Host Processor starts the `remoteproc` framework to manage the External System Processor.

1. Stop the External System Processor with the following command:

```
echo stop > /sys/class/remoteproc/remoteproc0/state
```

2. Start the External System Processor with the following command:

```
echo start > /sys/class/remoteproc/remoteproc0/state
```

1.3.9 Symmetric Multiprocessing

Warning: Symmetric multiprocessing (SMP) mode is only supported on Corstone-1000 with Cortex-A35 FVP but is disabled by default.

1. Build the software stack with SMP mode enabled:

```
kas build meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml:meta-arm/
↪kas/corstone1000-fvp-multicore.yml
```

2. Run the Corstone-1000 FVP:

```
kas shell meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml:meta-arm/
↪kas/corstone1000-fvp-multicore.yml \
-c "../meta-arm/scripts/runfvp"
```

3. Verify that the FVP is running the Host Processor with more than one CPU core:

```
nproc
4                   # number of processing units
```

1.3.10 Ethos-U85 NPU

Warning: The Ethos-U85 NPU is only supported on Corstone-1000 with Cortex-A320 FVP.

1. Clone the *systemready-patch* repository to your `${WORKSPACE}`.

```
cd ${WORKSPACE}
git clone https://git.gitlab.arm.com/arm-reference-solutions/systemready-
↪patch.git \
-b CORSTONE1000-2025.12
```

2. Copy the additional kas configuration file to:

```
cp ${WORKSPACE}/systemready-patch/embedded-a/corstone1000/ethos-u85_test/
↪ethos-u85_test.yml \
${WORKSPACE}/meta-arm/kas/
```

3. Re-Build the Corstone-1000 with Cortex-A320 FVP software stack as follows:

```
kas build meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml:meta-arm/
↪kas/corstone1000-a320.yml:\
meta-arm/kas/ethos-u85_test.yml
```

4. Run the Corstone-1000 with Cortex-320 FVP:

```
kas shell meta-arm/kas/corstone1000-fvp.yml:meta-arm/ci/debug.yml:meta-arm/
↪kas/corstone1000-a320.yml:\
systemready-patch/embedded-a/corstone1000/ethos-u85_test/ethos-u85_test.yml␣
↪\
-c "../meta-arm/scripts/runfvp"
```

5. To verify you are running the Corstone-1000 with Cortex-A320, build and run the FVP and inspect the CPU model reported in `/proc/cpuinfo` as shown below. Inside the FVP shell, confirm the core type:

```
grep -E 'CPU part|model name' /proc/cpuinfo
# Expect: CPU part : 0xd8f  (which corresponds to Cortex-A320)
```

6. Run the *delegate_runner* test application inside the FVP shell as follows:

```
delegate_runner -l /usr/lib/libethosu_op_delegate.so \
-n /usr/share/ethosu/mobilenet_v2_1.0_224_INT8_vela.tflite \
-i /usr/share/ethosu/input_data0.bin \
-o /usr/share/ethosu/actual_output_data0.bin
```

   The test completes in approximately one minute.

7. Run the following command to compare the generated output binary with the expected output binary:

```
cmp -s /usr/share/ethosu/expected_output_data0.bin /usr/share/ethosu/actual_
↪output_data0.bin
```

   The two binary files should be identical.

1.3.11 Secure Debug

> **Warning:** Secure Debug is only supported on MPS3.

The MPS3 supports Authenticated Debug Access Control (ADAC), using the CoreSight SDC-600 IP.

For more information about this, see the following resources:

- CoreSight SDC-600
- Authenticated Debug Access Control Specification
- Arm Corstone-1000 for MPS3 Application Note AN550, Chapter 7

The Secure Debug Manager API is implemented in the Secure Debug Manager (PSA-ADAC / SDC-600) repository. This repository also contains the necessary files for the Arm Development Studio support. The build and integration instructions can be found in its README.

The *Secure Debug Manager (PSA-ADAC / SDC-600)* repository also contains the private key and chain certificate to be used during the tests. The private key's public pair is provisioned into the One-Time Programmable memory in TrustedFirmware-M. These are dummy keys that should not be used in production.

To test the Secure Debug feature, you'll need a debug probe from the Arm ULINKpro family and Arm Development Studio versions 2022.2, 2022.c, or 2023.a.

1. Clone the *Secure Debug Manager (PSA-ADAC / SDC-600)* repository to your workspace.

   ```
   cd ${WORKSPACE}
   git clone https://github.com/ARM-software/secure-debug-manager.git
   ```

2. Navigate into the repository directory and checkout the specific commit in the listing below.

   ```
   cd ${WORKSPACE}/secure-debug-manager
   git checkout b30d6496ca749123e86b39b161b9f70ef76106d6
   ```

3. Follow the instructions in the Secure Debug Manager (PSA-ADAC / SDC-600)'s README for the development machine setup.

4. Rebuild the software stack with Secure Debug.

   ```
   kas build meta-arm/kas/corstone1000-mps3.yml:meta-arm/ci/debug.yml:meta-arm/
   ↪ci/secure-debug.yml
   ```

5. Flash the firmware image as shown *here*.

6. Run the software as shown *here*.

7. Wait until the Secure Enclave terminal (`ttyUSB1`) prints the following prompts:

   ```
   IComPortInit                  :  382 : warn  : init       : IComPortInit:␣
   ↪Blocked reading of LPH2RA is active.
   IComPortInit                  :  383 : warn  : init       : IComPortInit:␣
   ↪Blocked reading LPH2RA
   ```

8. Connect the debug probe to the MPS3 using the 20-pin 1.27mm connector with the `CS_20W_1.27MM silkscreen` label.

9. Create a debug configuration in Arm Development Studio as described in the Secure Debug Manager (PSA-ADAC / SDC-600)'s README.

10. Connect the debuger to the target using the debug configuration.
11. Provide the paths to the private key and trust chain certificate when asked by Arm Development Studio Console.

```
...

Please provide private key file path:
Enter file path > ${WORKSPACE}\secure-debug-manager\example\data\keys\
↪EcdsaP256Key-3.pem

Please provide trust chain file path:
Enter file path > ${WORKSPACE}\secure-debug-manager\example\data\chains\
↪chain.EcdsaP256-3

...
```

12. When successful authenticated, Arm Development Studio will connect to the running MPS3 and the debug features can be used. The following prompt should appear in the Secure Enclave terminal (`ttyUSB1`):

```
...
boot_platform_init: Corstone-1000 Secure Debug is a success.
...
```

1.4 Release notes

1.4.1 Disclaimer

You expressly assume all liabilities and risks relating to your use or operation of Your Software and Your Hardware designed or modified using the Arm Tools, including without limitation, Your software or Your Hardware designed or intended for safety-critical applications. Should Your Software or Your Hardware prove defective, you assume the entire cost of all necessary servicing, repair or correction.

1.4.2 Release notes - 2025.12

The same notes as the 2025.05 release still apply.

Known Issues or Limitations

- Corstone-1000 with Cortex-A320 FVP does not currently support Symmetric Multiprocessing
- Corstone-1000 with Cortex-A320 FVP becomes unresponsive when the Linux kernel driver for the Ethos-U85 NPU loads automatically after a software reboot.

1.4.3 Release notes - 2025.05

Known Issues or Limitations

- Crypto isolation is not supported in the Secure world of Corstone-1000. Additionally, clients in the Normal world are not isolated from one another.Therefore, if an end user wants to add a new Secure Partition (SP) (such as a software TPM) that accesses the Crypto service via the SE-Proxy, they are responsible for implementing their own isolation mechanisms to ensure proper security boundaries.
- DSTREAM debug probe may experience unreliable USB connectivity when used with Arm DS for secure debug. This issue is under active investigation, and we are working to identify and resolve compatibility issues in a future update. As a more stable alternative, the ULINKpro debug probe is recommended for use with Corstone-1000 in secure debug scenarios.

1.4.4 Release notes - 2024.11

The same notes as the 2024.06 release still apply.

1.4.5 Release notes - 2024.06

Known Issues or Limitations

- Use Ethernet over VirtIO due to lan91c111 Ethernet driver support dropped from U-Boot.
- Due to the performance uplimit of MPS3 FPGA and FVP, some Linux distros like Fedora Rawhide can not boot on Corstone-1000 (i.e. user may experience timeouts or boot hang).
- Corstone-1000 SoC on FVP doesn't have a secure debug peripheral. It does on the MPS3.
- See previous release notes for the known limitations regarding ACS tests.

Platform Support

- This software release is tested on Corstone-1000 FPGA version AN550_v2 https://developer.arm.com/downloads/-/download-fpga-images
- This software release is tested on Corstone-1000 Fast Model platform (FVP) version 11.23_25 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

SystemReady IR v2.0 Certification Milestone

As of this release, Corstone-1000 has achieved SystemReady IR v2.0 certification. This milestone confirms compliance with the SystemReady IR requirements, ensuring broader compatibility and reliability for deployment.

Applied patch 313ad2a0e600 to address compatibility requirements for SystemReady IR v2.0.

This update is included in tag CORSTONE1000-2024.06-systemready-ir-v2.0 and builds on the *CORSTONE1000-2024.06* release.

1.4.6 Release notes - 2023.11

Known Issues or Limitations

- Use Ethernet over VirtIO due to lan91c111 Ethernet driver support dropped from U-Boot.
- Temporally removing the External system support in Linux due to it using multiple custom devicetree bindings that caused problems with SystemReady IR 2.0 certification. For External system support please refer to the version 2023.06. We are aiming to restore it in a more standardised manner in our next release.
- Due to the performance uplimit of MPS3 FPGA and FVP, some Linux distros like Fedora Rawhide can not boot on Corstone-1000 (i.e. user may experience timeouts or boot hang).
- PSA Crypto tests (psa-crypto-api-test command) approximately take 30 minutes to complete for FVP and MPS3.
- Corstone-1000 SoC on FVP doesn't have a secure debug peripheral. It does on the MPS3.
- See previous release notes for the known limitations regarding ACS tests.

Platform Support

- This software release is tested on Corstone-1000 FPGA version AN550_v2 https://developer.arm.com/downloads/-/download-fpga-images
- This software release is tested on Corstone-1000 Fast Model platform (FVP) version 11.23_25 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

1.4.7 Release notes - 2023.06

Known Issues or Limitations

- FPGA supports Linux distro install and boot through installer. However, FVP only supports openSUSE raw image installation and boot.
- Due to the performance uplimit of MPS3 FPGA and FVP, some Linux distros like Fedora Rawhide can not boot on Corstone-1000 (i.e. user may experience timeouts or boot hang).
- PSA Crypto tests (psa-crypto-api-test command) take 30 minutes to complete for FVP and 1 hour for MPS3.
- Corstone-1000 SoC on FVP doesn't have a secure debug peripheral. It does on the MPS3 .
- The following limitations listed in the previous release are still applicable:
  - UEFI Compliant - Boot from network protocols must be implemented – FAILURE
  - Known limitations regarding ACS tests - see previous release's notes.

Platform Support

- This software release is tested on Corstone-1000 FPGA version AN550_v2 https://developer.arm.com/downloads/-/download-fpga-images
- This software release is tested on Corstone-1000 Fast Model platform (FVP) version 11.19_21 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

1.4.8 Release notes - 2022.11.23

Known Issues or Limitations

- The external-system can not be reset individually on (or using) AN550_v1 FPGA release. However, the system-wide reset still applies to the external-system.
- FPGA supports Linux distro install and boot through installer. However, FVP only supports openSUSE raw image installation and boot.
- Due to the performance uplimit of MPS3 FPGA and FVP, some Linux distros like Fedora Rawhide can not boot on Corstone-1000 (i.e. user may experience timeouts or boot hang).
- Below SCT FAILURE is a known issues in the FVP: UEFI Compliant - Boot from network protocols must be implemented – FAILURE
- Below SCT FAILURE is a known issue when a terminal emulator (in the system where the user connects to serial ports) does not support 80x25 or 80x50 mode: EFI_SIMPLE_TEXT_OUT_PROTOCOL.SetMode - SetMode() with valid mode – FAILURE
- Known limitations regarding ACS tests: The behavior after running ACS tests on FVP is not consistent. Both behaviors are expected and are valid; The system might boot till the Linux prompt. Or, the system might wait after finishing the ACS tests. In both cases, the system executes the entire test suite and writes the results as stated in the user guide.

Platform Support

- This software release is tested on Corstone-1000 FPGA version AN550_v1 https://developer.arm.com/downloads/-/download-fpga-images
- This software release is tested on Corstone-1000 Fast Model platform (FVP) version 11.19_21 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

1.4.9 Release notes - 2022.04.04

Known Issues or Limitations

- FPGA support Linux distro install and boot through installer. However, FVP only support openSUSE raw image installation and boot.
- Due to the performance uplimit of MPS3 FPGA and FVP, some Linux distros like Fedora Rawhide cannot boot on Corstone-1000 (i.e. user may experience timeouts or boot hang).
- Below SCT FAILURE is a known issues in the FVP: UEFI Compliant - Boot from network protocols must be implemented – FAILURE

Platform Support

- This software release is tested on Corstone-1000 FPGA version AN550_v1
- This software release is tested on Corstone-1000 Fast Model platform (FVP) version 11.17_23 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

1.4.10 Release notes - 2022.02.25

Known Issues or Limitations

- The following tests only work on Corstone-1000 FPGA: ACS tests (SCT, FWTS, BSA), manual capsule update test, Linux distro install and boot.

Platform Support

- This software release is tested on Corstone-1000 FPGA version AN550_v1
- This software release is tested on Corstone-1000 Fast Model platform (FVP) version 11.17_23 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

Release notes - 2022.02.21

Known Issues or Limitations

- The following tests only work on Corstone-1000 FPGA: ACS tests (SCT, FWTS, BSA), manual capsule update test, Linux distro install and boot, psa-arch-test.

Platform Support

- This software release is tested on Corstone-1000 FPGA version AN550_v1
- This software release is tested on Corstone-1000 Fast Model platform (FVP) version 11.16.21 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

Release notes - 2022.01.18

Known Issues or Limitations

- Before running each SystemReady-IR tests: ACS tests (SCT, FWTS, BSA), manual capsule update test, Linux distro install and boot, etc., the SecureEnclave flash must be cleaned. See user-guide “Clean Secure Flash Before Testing” section.

Release notes - 2021.12.15

Software Features

The following components are present in the release:

- Yocto version Honister
- Linux kernel version 5.10
- U-Boot 2021.07
- OP-TEE version 3.14
- Trusted Firmware-A 2.5
- Trusted Firmware-M 1.5
- OpenAMP 347397decaa43372fc4d00f965640ebde042966d
- Trusted Services a365a04f937b9b76ebb2e0eeade226f208cbc0d2

Platform Support

- This software release is tested on Corstone-1000 FPGA version AN550_v1
- This software release is tested on Corstone-1000 Fast Model platform (FVP) version 11.16.21 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

Known Issues or Limitations

- The following tests only work on Corstone-1000 FPGA: ACS tests (SCT, FWTS, BSA), manual capsule update test, Linux distro install and boot, and psa-arch-tests.
- Only the manual capsule update from UEFI shell is supported on FPGA.
- Due to flash size limitation and to support A/B banks,the wic image provided by the user should be smaller than 15MB.
- The failures in PSA Arch Crypto Test are known limitations with crypto library. It requires further investigation. The user can refer to PSA Arch Crypto Test Failure Analysis In TF-M V1.5 Release for the reason for each failing test.

Release notes - 2021.10.29

Software Features

This initial release of Corstone-1000 supports booting Linux on the Cortex-A35 and TF-M/MCUBOOT in the Secure Enclave. The following components are present in the release:

- Linux kernel version 5.10
- U-Boot 2021.07
- OP-TEE version 3.14
- Trusted Firmware-A 2.5
- Trusted Firmware-M 1.4

Platform Support

- This Software release is tested on Corstone-1000 Fast Model platform (FVP) version 11.16.21 https://developer.arm.com/tools-and-software/open-source-software/arm-platforms-software/arm-ecosystem-fvps

Known Issues or Limitations

- No software support for external system(Cortex M3)
- No communication established between A35 and M0+
- Very basic functionality of booting Secure Enclave, Trusted Firmware-A , OP-TEE , u-boot and Linux are performed

Support

For technical support email: support-subsystem-iot@arm.com

For all security issues, contact Arm by email at psirt@arm.com.

1.5 Change Log

This document contains a summary of the new features, changes and fixes in each release of Corstone-1000 software stack.

1.5.1 Version 2025.12

Changes

- Delivered end-to-end Cortex-A320 enablement across U-Boot, TF-A, TF-M, OP-TEE, Yocto machine layers, and documentation, including device-tree updates, MPIDR handling, and FVP model renaming.
- Rolled out the PSA Firmware Update (DEN0118) pipeline: U-Boot capsule parsing, Bootloader Abstraction Layer in TF-M, ESRT exposure, and Trusted Services IPC bridges replacing legacy capsule code.
- Hardened the new firmware update flow with EFI self-tests, metadata restructuring for partial and multi-image acceptance, and RSE-COMMS gating refinements.
- Upgraded key firmware components (TF-A 2.13.0, TF-M 2.2.1, Trusted Services 1.2.0, OP-TEE OS 4.7.0) and introduced targeted test skips plus integer-only build modes to keep validation green.
- Cleaned and renumbered downstream patch series across Trusted Services and TF-M while removing obsolete integrations to align with upstream baselines.
- Refreshed release material and architecture guides to describe the A320 profile, PSA FWU behavior, and updated software stack.
- Added KAS profiles, machine includes, and automated FVP selection logic to streamline developer workflows for the refreshed platform configuration.

Corstone-1000 components versions

linux-yocto	6.12.60
u-boot	2025.04
external-system	0.1.0
optee-client	4.7.0
optee-os	4.7.0
trusted-firmware-a	2.13.0
trusted-firmware-m	2.2.1
libts	v1.2.0
ts-sp-{se-proxy, smm-gateway}	v1.2.0
ts-psa-{crypto, iat, its. ps}-api-test	74dc6646ff

Yocto distribution components versions

meta-arm	whinlatter
bitbake	0dde1a3ff8
meta-openembedded	fc0152e434
openembedded-core	4bd920ad7d
meta-yocto	b3b6592635
meta-secure-core	63209fb150
meta-ethos	aa2504a32f
meta-sca	e68f1a9d17
busybox	1.37.0
musl	1.2.5
gcc-arm-none-eabi	13.3.rel1
gcc-cross-aarch64	15.2.0
openssl	3.5.4

1.5.2 Version 2025.05

Changes

- OP-TEE OS: Added support for v4.4
- Trusted Services: PSA-Crypto structures aligned with TF-M, added protobuf interface to crypto-sp
- Documentation: fixed typos, added host-level authentication section, enabled fly-out sidebar menu
- Das U-Boot: Reserved memory for RSS communication-pointer access protocol
- Linux Kernel: Upgraded kernel to v6.12, updated Upstream-Status notes for remoteproc patches
- Corstone-1000 image: Implemented IMAGE_ROOTFS_EXTRA_SPACE workaround

Corstone-1000 components versions

linux-yocto	6.12.30
u-boot	2023.07.02
external-system	0.1.0
optee-client	4.4.0
optee-os	4.4.0
trusted-firmware-a	2.11.0
trusted-firmware-m	2.1.1
libts	602be60719
ts-newlib	4.1.0
ts-psa-{crypto, iat, its. ps}-api-test	74dc6646ff
ts-sp-{se-proxy, smm-gateway}	602be60719

Yocto distribution components versions

meta-arm	walnascar
poky	ee0d8d8a61
meta-openembedded	2169c9afcc
meta-secure-core	423bc85b05
busybox	1.37.0
musl	1.2.5
gcc-arm-none-eabi	13.3.rel1
gcc-cross-aarch64	14.2.0
openssl	3.4.1

1.5.3 Version 2024.11

Changes

- Implementation of a replication strategy for FWU metadata in TF-M according to the FWU specification.
- Upgrade to metadata version 2 in TF-M.
- Increase the ITS and PS memory size in Secure Flash for TF-M.
- SW components upgrades.
- Bug fixes.

Corstone-1000 components versions

linux-yocto	6.10.14
u-boot	2023.07.02
external-system	0.1.0
optee-client	4.2.0
optee-os	4.2.0
trusted-firmware-a	2.11.0
trusted-firmware-m	2.1.0
libts	602be60719
ts-newlib	4.1.0
ts-psa-{crypto, iat, its. ps}-api-test	74dc6646ff
ts-sp-{se-proxy, smm-gateway}	602be60719

Yocto distribution components versions

meta-arm	styhead
poky	5465094be9
meta-openembedded	461d85a183
meta-secure-core	59d7e90542
busybox	1.36.1
musl	1.2.5
gcc-arm-none-eabi	13.3.rel1
gcc-cross-aarch64	14.2.0
openssl	3.3.1

1.5.4 Version 2024.06

Changes

- Re-enabling support for the External System using linux remoteproc (only supporting switching on and off the External System)
- UEFI Secure Boot and Authenticated Variable support
- RSE Comms replaces OpenAMP
- The EFI System partition image is now created by the meta-arm build system. This image is mounted on the second MMC card by default in the FVP.
- The capsule generation script is now part of the meta-arm build system. Corstone1000-flash-firmware-image recipe generates a capsule binary using the U-Boot capsule generation tool that includes all the firmware binaries and recovery kernel image.
- SW components upgrades
- Bug fixes

Corstone-1000 components versions

arm-tstee	2.0.0
linux-yocto	6.6.23
u-boot	2023.07.02
external-system	0.1.0
optee-client	4.1.0
optee-os	4.1.0
trusted-firmware-a	2.10.4
trusted-firmware-m	2.0.0
libts	602be60719
ts-newlib	4.1.0
ts-psa-{crypto, iat, its. ps}-api-test	602be60719
ts-sp-{se-proxy, smm-gateway}	602be60719

Yocto distribution components versions

meta-arm	scarthgap
poky	scarthgap
meta-openembedded	scarthgap
meta-secure-core	scarthgap
busybox	1.36.1
musl	1.2.4
gcc-arm-none-eabi	13.2.Rel1
gcc-cross-aarch64	13.2.0
openssl	3.2.1

1.5.5 Version 2023.11

Changes

- Making Corstone-1000 SystemReady IR 2.0 certifiable
- Allow booting Debian & OpenSUSE on FVP
- Add support for two MMC cards for the FVP
- Add signed capsule update support
- Enable on-disk capsule update
- Add the feature of purging specific DT nodes in U-Boot before Linux
- Add Ethernet over VirtIO support in U-Boot
- Add support for unaligned MMC card images
- Reducing the out-of-tree patches by upstreaming them to the corresponding open-source projects
- SW components upgrades
- Bug fixes

Corstone-1000 components versions

arm-ffa-tee	1.1.2-r0
linux-yocto	6.5.7
u-boot	2023.07
external-system	0.1.0+gitAUTOINC+8c9dca74b1-r0
optee-client	3.22.0
optee-os	3.22.0
trusted-firmware-a	2.9.0
trusted-firmware-m	1.8.1
libts	08b3d39471
ts-newlib	4.1.0
ts-psa-{crypto, iat, its. ps}-api-test	38cb53a4d9
ts-sp-{se-proxy, smm-gateway}	08b3d39471

Yocto distribution components versions

meta-arm	nanbield
poky	nanbield
meta-openembedded	nanbield
meta-secure-core	nanbield
busybox	1.36.1
musl	1.2.4
gcc-arm-none-eabi	11.2-2022.02
gcc-cross-aarch64	13.2.0
openssl	3.1.3

1.5.6 Version 2023.06

Changes

- GPT support (in TF-M, TF-A, U-boot)
- Use TF-M BL1 code as the ROM code instead of MCUboot (the next stage bootloader BL2 remains to be MCUboot)
- Secure Enclave uses CC312 OTP as the provisioning backend in FVP and FPGA
- NVMXIP block storage support in U-Boot
- Upgrading the SW stack recipes
- Upgrades for the U-Boot FF-A driver and MM communication

Corstone-1000 components versions

arm-ffa-tee	1.1.2-r0
arm-ffa-user	5.0.1-r0
corstone1000-external-sys-tests	1.0+gitAUTOINC+2945cd92f7-r0
external-system	0.1.0+gitAUTOINC+8c9dca74b1-r0
linux-yocto	6.1.25+gitAUTOINC+36901b5b29_581dc1aa2f-r0
u-boot	2023.01-r0
optee-client	3.18.0-r0
optee-os	3.20.0-r0
trusted-firmware-a	2.8.0-r0
trusted-firmware-m	1.7.0-r0
ts-newlib	4.1.0-r0
ts-psa-{crypto, iat, its. ps}-api-test	38cb53a4d9
ts-sp-{se-proxy, smm-gateway}	08b3d39471

Yocto distribution components versions

meta-arm	mickledore
poky	mickledore
meta-openembedded	mickledore
busybox	1.36.0-r0
musl	1.2.3+gitAUTOINC+7d756e1c04-r0
gcc-arm-none-eabi-native	11.2-2022.02
gcc-cross-aarch64	12.2.rel1-r0
openssl	3.1.0-r0

1.5.7 Version 2022.11.23

Changes

- Booting the External System (Cortex-M3) with RTX RTOS
- Adding MHU communication between the HOST (Cortex-A35) and the External System
- Adding a Linux application to test the External System
- Adding ESRT (EFI System Resource Table) support
- Upgrading the SW stack recipes
- Upgrades for the U-Boot FF-A driver and MM communication

Corstone-1000 components versions

arm-ffa-tee	1.1.1
arm-ffa-user	5.0.0
corstone1000-external-sys-tests	1.0
external-system	0.1.0
linux-yocto	5.19
u-boot	2022.07
optee-client	3.18.0
optee-os	3.18.0
trusted-firmware-a	2.7.0
trusted-firmware-m	1.6.0
ts-newlib	4.1.0
ts-psa-{crypto, iat, its. ps}-api-test	451aa087a4
ts-sp-{se-proxy, smm-gateway}	3d4956770f

Yocto distribution components versions

meta-arm	langdale
poky	langdale
meta-openembedded	langdale
busybox	1.35.0
musl	1.2.3+git37e18b7bf3
gcc-arm-none-eabi-native	11.2-2022.02
gcc-cross-aarch64	12.2
openssl	3.0.5

1.5.8 Version 2022.04.04

Changes

- Linux distro openSUSE, raw image installation and boot in the FVP.
- SCT test support in FVP.
- Manual capsule update support in FVP.

1.5.9 Version 2022.02.25

Changes

- Building and running psa-arch-tests on Corstone-1000 FVP
- Enabled smm-gateway partition in Trusted Service on Corstone-1000 FVP
- Enabled MHU driver in Trusted Service on Corstone-1000 FVP
- Enabled OpenAMP support in SE proxy SP on Corstone-1000 FVP

1.5.10 Version 2022.02.21

Changes

- psa-arch-tests: recipe is dropped and merged into the secure-partitons recipe.
- psa-arch-tests: The tests are align with latest tfm version for psa-crypto-api suite.

1.5.11 Version 2022.01.18

Changes

- psa-arch-tests: change master to main for psa-arch-tests
- U-Boot: fix null pointer exception for get_image_info
- TF-M: fix capsule instability issue for Corstone-1000

1.5.12 Version 2022.01.07

Changes

- Corstone-1000: fix SystemReady-IR ACS test (SCT, FWTS) failures.
- U-Boot: send bootcomplete event to secure enclave.
- U-Boot: support populating Corstone-1000 image_info to ESRT table.
- U-Boot: add ethernet device and enable configs to support bootfromnetwork SCT.

1.5.13 Version 2021.12.15

Changes

- Enabling Corstone-1000 FPGA support on: - Linux 5.10 - OP-TEE 3.14 - Trusted Firmware-A 2.5 - Trusted Firmware-M 1.5
- Building and running psa-arch-tests
- Adding openamp support in SE proxy SP
- OP-TEE: adding smm-gateway partition
- U-Boot: introducing Arm FF-A and MM support

1.5.14 Version 2021.10.29

Changes

- Enabling Corstone-1000 FVP support on: - Linux 5.10 - OP-TEE 3.14 - Trusted Firmware-A 2.5 - Trusted Firmware-M 1.4
- Linux kernel: enabling EFI, adding FF-A debugfs driver, integrating ARM_FFA_TRANSPORT.
- U-Boot: Extending EFI support

- python3-imgtool: adding recipe for Trusted-firmware-m
- python3-imgtool: adding the Yocto recipe used in signing host images (based on MCUBOOT format)
