TF-A overview

Contents

Stable: 22.04.2021 - 11:20 / Revision: 30.04.2021 - 10:17

A quality version of this page, approved on *22 April 2021*, was based off this revision.

Contents

1 Trusted Firmware-A

Trusted Firmware-A is a reference implementation of secure-world software provided by Arm®. It was first designed for Armv8-A platforms, and has been adapted to be used on Armv7-A platforms by STMicroelectronics. Trusted Firmware-A is part of the Trusted Firmware project that is an open governance community project hosted by Linaro.[1]

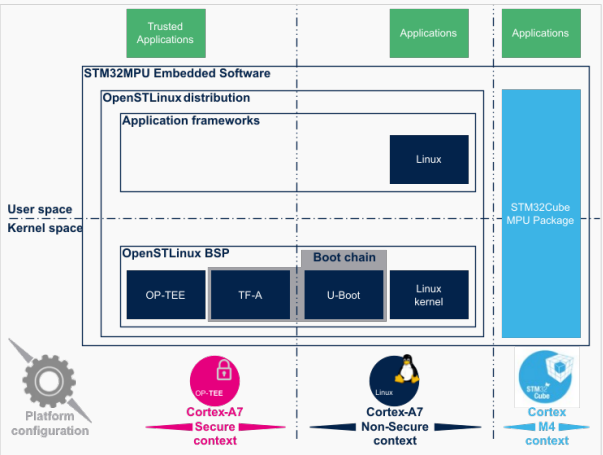
It is used as the first-stage boot loader (FSBL) on STM32 MPU platforms when using the trusted boot chain.

The code is open source, under a BSD-3-Clause license, and can be found on Linaro project page [2], including an up-to-date documentation about Trusted Firmware-A implementation [3].

Trusted Firmware-A also implements a set of features with various Arm interface standards:

- The power state coordination interface (PSCI) [4]
- SMC calling convention [5]
- System control and management interface [6]

Trusted Firmware-A is usually shortened to TF-A.



Zoom out to STM32MPU Embedded Software

2 Architecture

The global architecture of TF-A is explained in the Trusted Firmware-A design [7] document.

TF-A is divided into several binaries, each with a dedicated main role.
For 32-bit Arm processors (AArch32), the trusted boot is divided into four stages (in order of execution):

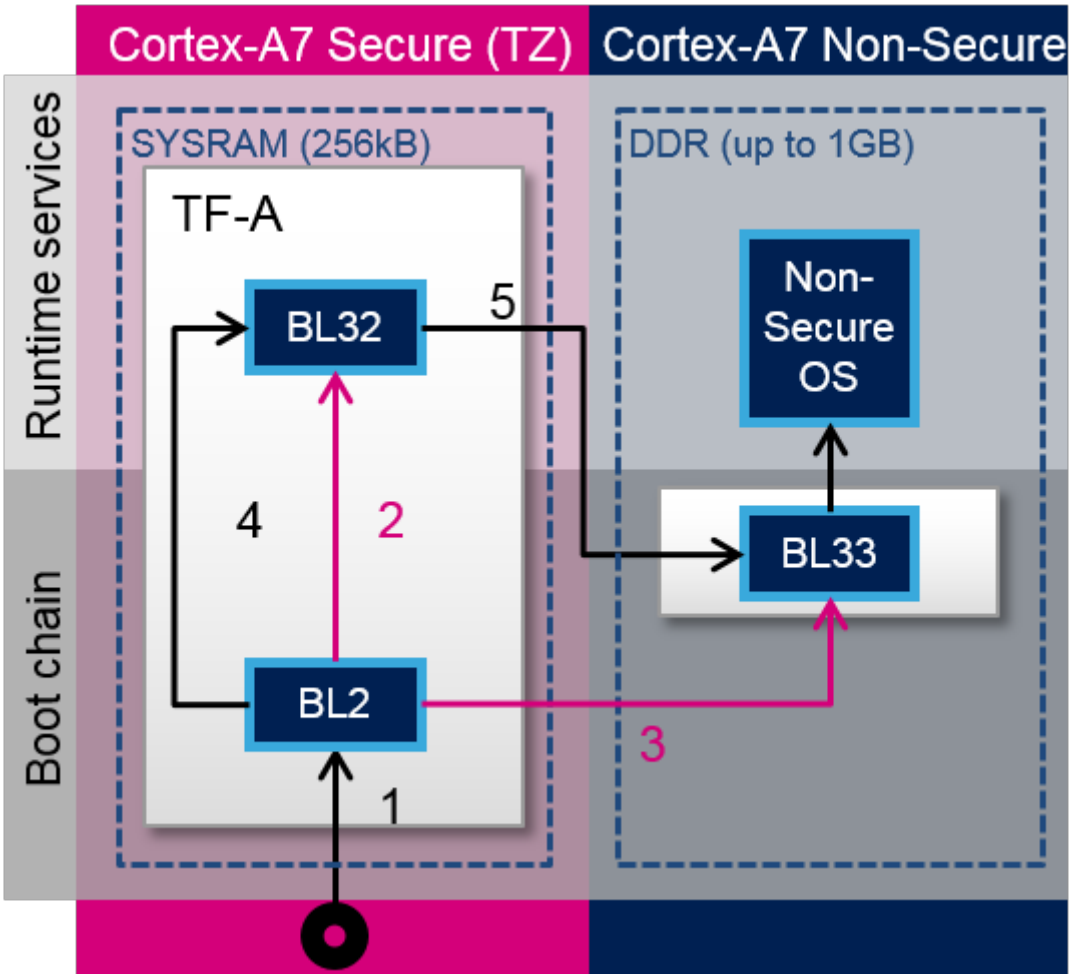
- Boot loader stage 1 (BL1) application processor trusted ROM
- Boot loader stage 2 (BL2) trusted boot firmware
- Boot loader stage 3-2 (BL32) runtime software
- Boot loader stage 3-3 (BL33) non-trusted firmware

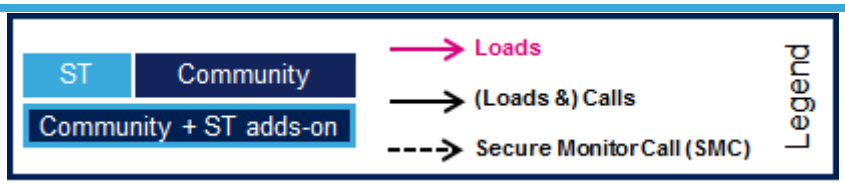
BL1, BL2 and BL32 are parts of TF-A.

Because STM32 MPU platforms uses a dedicated ROM code, the BL1 boot stage is then removed. ROM code expects the BL2 to run at EL3 execution level. This mode is selected when the BL2_AT_EL3 build flag is enabled.

BL33 is outside of TF-A. This is the first non-secure code loaded by TF-A. During the boot sequence, this is the secondary stage boot loader (SSBL). For STM32 MPU platforms, the SSBL is U-Boot by default.

TF-A can manage its configuration with a device tree. In the BL2 stage, it is a reduced version of the Linux kernel one, with only the required devices used during boot. It can be configured with STM32CubeMX.





TF-A loading steps:

1. ROM code loads and calls BL2
2. BL2 loads BL32
3. BL2 loads BL33
4. BL2 calls BL32
5. BL32 calls BL33

3 Boot loader stages

3.1 BL1

BL1 is the first stage executed, and is designed to act as ROM code; it is loaded and executed in internal RAM. It is not used for the STM32 MPU. As the STM32 MPU has its own proprietary ROM code, this part can be removed and BL2 is then the first TF-A binary to be executed.

3.2 BL2

BL2 is in charge of loading the next-stage images (secure and non secure). To achieve this role, BL2 has to initialize all the required peripherals.

- System components: clocks, DDR, ...
- Security components: Firewall
- Storage

BL2 offers different features to load and authenticate images.

At the end of its execution, after having loaded BL32 and the next boot stage (BL33), BL2 jumps to BL32.

3.2.1 FIP

The Firmware Image Package (FIP)[8] is a TF-A archive binary that encapsulates bootloader images into a single archive. It can also contain other data such as certificates that are required to complete the boot process. A dedicated driver drivers/io/io_fip.c able to read data from this package is part of the TF-A BL2.
FIP uses a specific layout based on a table of contents followed by payload data. It is synchronized between the driver and the host creation tool: Fiptool tools/fiptool/fiptool.c . This host tool is able to create a package, get info from the package, update, unpack or remove data in this package.

3.2.2 Firmware Configuration

The Firmware Configuration Framework (FCONF)[9] is a way to offer more flexibility in the firmware. It is used to provide most of the platform-specific data that were previously hard coded inside the firmware. This framework uses device tree (one or multiple) that are passed to the firmware during load processing. BL2 uses it to describe the chain of trust and the images list to be loaded.
Thanks to device tree usage, the configuration becomes dynamic at boot time. The current implementation uses the following device tree as framework entry:

- FW_CONFIG - The firmware configuration file. Hold properties shared across all BLx images. An example is the **dtb-registry** node, which contains the information about other binaries configuration (load-address, size, image_id).
- HW_CONFIG - The hardware configuration file. Can be shared by all Boot Loader stages and also by the Normal World Rich OS.
- TB_FW_CONFIG - Trusted Boot Firmware configuration file. Shared between BL1 and BL2.
- SOC_FW_CONFIG - SoC Firmware configuration file. Used by BL31.
- TOS_FW_CONFIG - Trusted OS Firmware configuration file. Used by Trusted OS (BL32).
- NT_FW_CONFIG - Non Trusted Firmware configuration file. Used by Non-trusted firmware (BL33).

3.2.3 Authentication

TF-A BL2 implements an authentication framework that uses a defined Chain of Trust (CoT) based on Arm TBBR[10] requirement to achieve a secure boot. The authentication is enabled as soon as the TRUSTED_BOARD_BOOT flag is defined. TF-A BL2 implements this CoT which is based on a Root of Trust Public Key (ROTPK). The CoT relies on a public key infrastructure generating self-signed certificate (following X509 v3 standard [11]). There is **no Certificate Authority (CA)** because the CoT is not established by verifying the validity of a certificate's issuer.

Different keys are used for this CoT:

- Root of trust key - The private part of this key is used to sign the BL2 content certificate and the trusted key certificate. The public part is the ROTPK.
- Trusted world key - The private part is used to sign the key certificates corresponding to the secure world images (SCP_BL2, BL31 and BL32). The public part is stored in one of the extension fields in the trusted world certificate.
- Non-trusted world key - The private part is used to sign the key certificate corresponding to the non secure world image (BL33). The public part is stored in one of the extension fields in the trusted world certificate.
- BL3X keys - For each of SCP_BL2, BL31, BL32 and BL33, the private part is used to sign the content certificate for the BL3X image. The public part is stored in one of the extension fields in the corresponding key certificate.

The certificates used in this CoT could be Key certificate or Content certificate.

- BL2 content certificate - It is self-signed with the private part of the ROT key. It contains a hash of the BL2 image.
- Trusted key certificate - It is self-signed with the private part of the ROT key. It contains the public part of the trusted world key and the public part of the non-trusted world key.
- SCP_BL2 key certificate - It is self-signed with the trusted world key. It contains the public part of the SCP_BL2 key.
- SCP_BL2 content certificate - It is self-signed with the SCP_BL2 key. It contains a hash of the SCP_BL2 image.
- BL31 key certificate - It is self-signed with the trusted world key. It contains the public part of the BL31 key.
- BL31 content certificate - It is self-signed with the BL31 key. It contains a hash of the BL31 image.
- BL32 key certificate - It is self-signed with the trusted world key. It contains the public part of the BL32 key.
- BL32 content certificate - It is self-signed with the BL32 key. It contains a hash of the BL32 image.
- BL33 key certificate - It is self-signed with the non-trusted world key. It contains the public part of the BL33 key.
- BL33 content certificate - It is self-signed with the BL33 key. It contains a hash of the BL33 image.

3.3 BL32

BL32 provides runtime secure services.

On Armv7 architecture, the BL32 must embed a Secure Monitor as it will be executed in the same privilege level (PL1-SVC Secure). TF-A provides a minimal monitor implementation: SP-MIN. It is described in the TF-A functionality list [3] as: "A minimal AArch32 Secure Payload (SP-MIN) to demonstrate PSCI [4] library integration with AArch32 EL3 Runtime Software."

This minimal implementation can be replaced with a trusted OS or trusted environment execution (TEE), such as OP-TEE that also embeds a secure monitor on Armv7. Both solutions (SP-MIN or OP-TEE) are supported by STMicroelectronics for STM32MP15.

BL32 acts as a secure monitor and thus provides secure services to non-secure OSs. These services are called by non-secure software with secure monitor calls [5].

This code is in charge of standard service calls, like PSCI [4] or SCMI [6].

It also provides STMicroelectronics proprietary services to access secure peripherals (with secure access control).

4 References

- https://www.trustedfirmware.org/
- https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git
- 3.03.1 https://trustedfirmware-a.readthedocs.io/en/latest/
- 4.04.14.2 ARM Power State Coordination Interface
- 5.05.1 SMC Calling Convention (SMCCC)
- 6.06.1 Arm System Control and Management Interface
- https://trustedfirmware-a.readthedocs.io/en/latest/design/index.html
- https://trustedfirmware-a.readthedocs.io/en/latest/design/firmware-design.html?highlight=FIP#firmware-image-package-fip
- https://trustedfirmware-a.readthedocs.io/en/latest/components/fconf/index.html?highlight=FCONF
- https://trustedfirmware-a.readthedocs.io/en/latest/design/trusted-board-boot.html
- https://tools.ietf.org/rfc/rfc5280.txt