ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

TZC internal peripheral

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Contents

1. TZC internal peripheral ... 3
2. How to assign an internal peripheral to a runtime context ... 9
3. OP-TEE overview ... 16
4. STM32CubeMX ... 24
5. STM32MP15 resources ... 27
6. STM32MPU Embedded Software architecture overview ... 31
7. TF-A overview ... 35

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

A quality version of this page, approved on *4 February 2020*, was based off this revision.

Contents

1 Article purpose 4
2 Peripheral overview 5
2.1 Features 5
2.2 Security support 5
3 Peripheral usage and associated software 6
3.1 Boot time 6
3.2 Runtime 6
3.2.1 Overview 6
3.2.2 Software frameworks 6
3.2.3 Peripheral configuration 6
3.2.4 Peripheral assignment 6
4 How to go further 8
5 References 9

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

1 Article purpose

The purpose of this article is to:

- briefly introduce the TZC peripheral and its main features
- indicate the level of security supported by this hardware block
- explain how it can be allocated to the three runtime contexts and linked to the corresponding software components
- explain, when necessary, how to configure the TZC peripheral.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

2 Peripheral overview

The TZC peripheral is used to filter read/write accesses to the DDR controller according to TrustZone access rights, and according to Non-Secure master Address ID (NSAID) on up to 9 programmable regions.

2.1 Features

Refer to the STM32MP15 reference manuals for the complete list of features, and to the software components, introduced below, to see which features are implemented.

2.2 Security support

The TZC is a **secure** peripheral.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

3 Peripheral usage and associated software

3.1 Boot time

The TZC is configured at boot time to setup DDR accesses.

3.2 Runtime

3.2.1 Overview

The TZC is a system peripheral and is controlled by the Arm®Cortex®-A7 secure.

3.2.2 Software frameworks

Domain	Peripheral	Software frameworks			Comment
Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4 (STM32Cube)			
Security	TZC	OP-TEE TZC driver			

3.2.3 Peripheral configuration

The configuration is applied by the firmware running in the secure context.

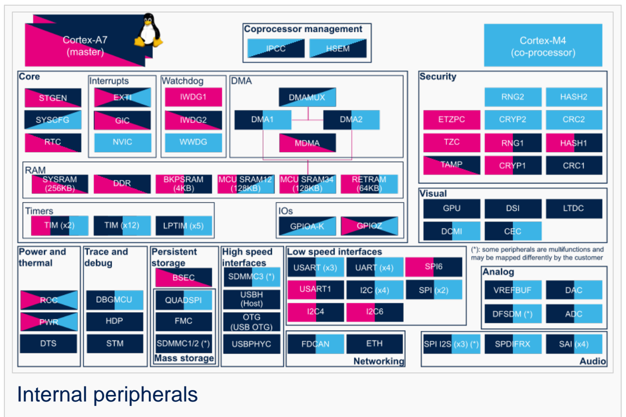
This configuration is done in TF-A or in OP-TEE.

3.2.4 Peripheral assignment

Check boxes illustrate the possible peripheral allocations supported by STM32 MPU Embedded Software:

- means that the peripheral can be assigned () to the given runtime context.
- is used for system peripherals that cannot be unchecked because they are statically connected in the device.

Refer to How to assign an internal peripheral to a runtime context for more information on how to assign peripherals manually or via STM32CubeMX.
The present chapter describes STMicroelectronics recommendations or choice of implementation. Additional possiblities might be described in STM32MP15 reference manuals
.



Internal peripherals

Domain	Peripheral	Runtime allocation			Comment
	Cortex-A7	Cortex-A7	Cortex-M4		

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

TZC internal peripheral

Domain	Peripheral	Runtime allocation				Comment
Instance	secure (OP-TEE)	non-secure (Linux)	(STM32Cube)			
Security	TZC	TZC				

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

4 How to go further

The TZC is an Arm® peripheral: TZC-400 TrustZone Address Space Controller[1]

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

5 References

- http://infocenter.arm.com/help/topic/com.arm.doc.ddi0504c/DDI0504C_tzc400_r0p1_trm.pdf

TrustZone® address space Controller for DDR

Arm® and TrustZone® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Doubledata rate (memory domain)

TrustZone®

Arm® and TrustZone® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

Open Portable Trusted Execution Environment

Linux® is a registered trademark of Linus Torvalds.

Stable: 16.02.2021 - 17:29 / Revision: 16.02.2021 - 17:11

A quality version of this page, approved on *16 February 2021*, was based off this revision.

Contents

1 Article purpose 10
2 Introduction 11
3 STM32CubeMX generated assignment 12
4 Manual assignment 14
4.1 TF-A 14
4.2 U-boot 14
4.3 Linux kernel 15
4.4 STM32Cube 15
4.5 OP-TEE 16

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

1 Article purpose

This article explains how to configure the software that assigns a peripheral to a runtime context.

2 Introduction

A peripheral can be **assigned** to a runtime context via the configuration defined in the device tree. The device tree can be either generated by the STM32CubeMX tool or edited manually.

On STM32MP15 line devices, the assignement can be strengthened by a hardware mechanism: the ETZPC internal peripheral, which is configured by the TF-A boot loader. The ETZPC internal peripheral isolates the peripherals for the Cortex-A7 secure or the Cortex-M4 context. The peripherals assigned to the Cortex-A7 non-secure context are visible from any context, without any isolation.

The components running on the platform after TF-A execution (such as U-Boot, Linux, STM32Cube and OP-TEE) must have a **configuration** that is consistent with the assignment and the isolation configurations.

The following sections describe how to configure TF-A, U-Boot, Linux and STM32Cube accordingly.

Information

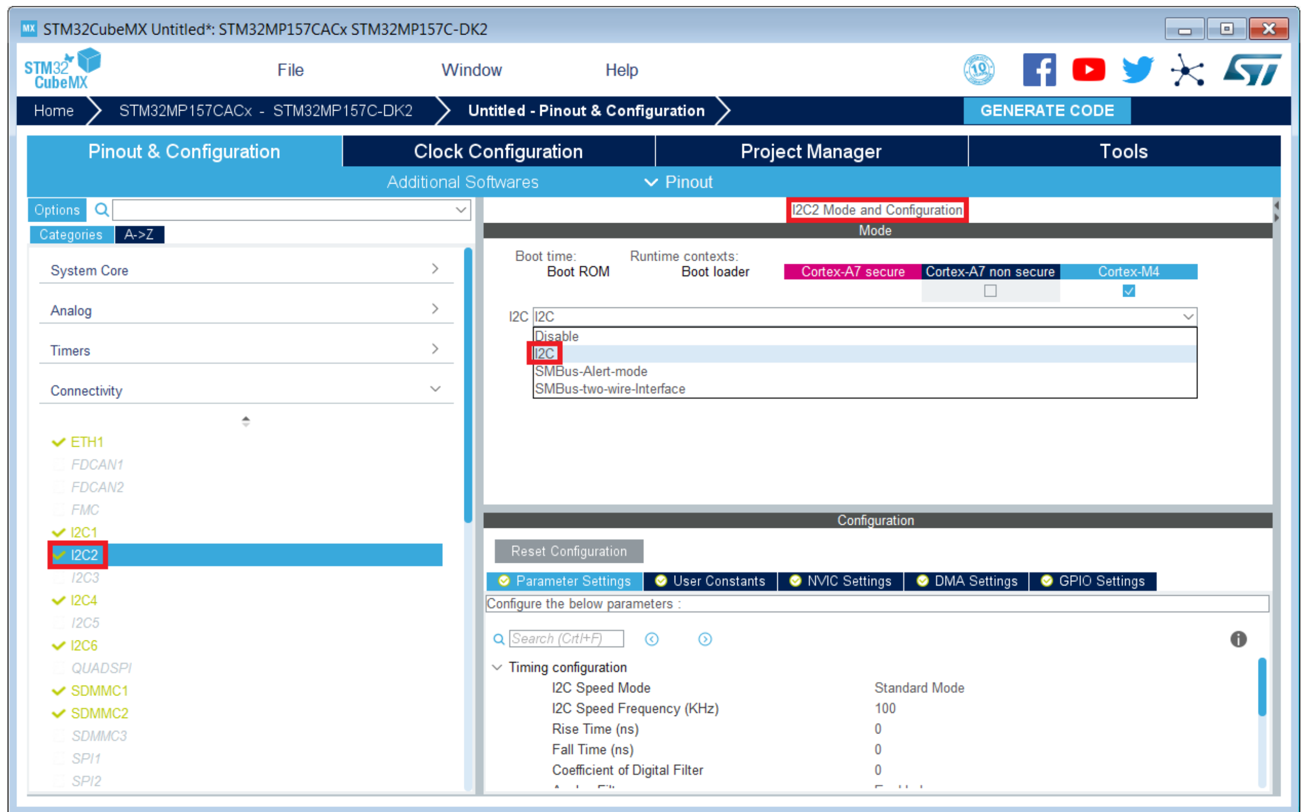
Beyond the peripherals assignment, explained in this article, it is also important to understand How to configure system resources (i.e clocks, regulator, gpio,...), shared between the Cortex-A7 and Cortex-M4 contexts

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

3 STM32CubeMX generated assignment

The screenshot below shows the STM32CubeMX user interface:

- **I2C2** peripheral is selected, on the left
- **I2C2 Mode and Configuration** panel, on the right, shows that this I2C instance can be assigned to the Cortex-A7 non-secure or the Cortex-M4 (that is selected) runtime context
- **I2C** mode is enabled in the drop down menu



The context assignment table is displayed inside each peripheral **Mode and Configuration** panel but it is possible to display it for all the peripherals in the **Options** menu via the **Show contexts** option

The **GENERATE CODE** button, on the top right, produces the following:

- The **TF-A device tree** with the ETZPC configuration that isolates the I2C2 instance (in the example) for the Cortex-M4 context. This same device tree can be used by **OP-TEE**, when enabled
- The **U-Boot device tree** widely inherited from the Linux one, just below
- The **Linux kernel device tree** with the I2C node disabled for Linux and enabled for the coprocessor
- The **STM32Cube project** with I2C2 HAL initialization code

The Manual assignment section, just below, illustrates what STM32CubeMX is generating as it follows the same example.

In addition of this generation, the user may have to manually complete the system resources configuration in the user sections embedded in the STM32CubeMX generated device tree. Refer to How to configure system resources for details.

4 Manual assignment

This section gives step by step instructions, per software components, to manually perform the peripherals assignments.
It takes the same I2C2 example as the previous section, that showed how to use STM32CubeMX, in order to make the move from one approach to the other easier.

Information

The assignments combinations described in the STM32MP15 peripherals overview article are naturally supported by STM32MPU Embedded Software distribution. Note that the STM32MP15 reference manual may describe more options that would require embedded software adaptations

4.1 TF-A

The assignment follows the ETZPC device tree configuration, with below possible values:

- **DECPROT_S_RW** for the Cortex-A7 secure (Secure OS like OP-TEE)
- **DECPROT_NS_RW** for the Cortex-A7 non-secure (Linux)
  - As stated earlier in this article, there is no hardware isolation for the Cortex-A7 non-secure so this value allows accesses from any context
- **DECPROT_MCU_ISOLATION** for the Cortex-M4 (STM32Cube)

Example:

```
@etzpc: etzpc@5C007000 {
       st,decprot = <
           DECPROT(STM32MP1_ETZPC_I2C2_ID, DECPROT_MCU_ISOLATION, DECPROT_UNLOCK)
       >;
};
```

Information

The value **DECPROT_NS_RW** can be used with **DECPROT_LOCK** as last parameter. In Cortex-M4 context, this specific configuration allows the generation of an error in the resource manager utility while trying to use on Cortex-M4 side a peripheral that is assigned to the Cortex-A7 non-secure context. If **DECPROT_UNLOCK** is used, then the utility allows the Cortex-M4 to use a peripheral that is assigned to the Cortex-A7 non-secure context.

4.2 U-boot

No specific configuration is needed in U-Boot to configure the access to the peripheral.

Information

U-Boot does not perform any check with regards to ETZPC configuration before accessing to a peripheral. In case of inconsistency an illegal access is generated.

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

Information

U-Boot checks the consistency between ETZPC isolation configuration and Linux kernel device tree configuration to guarantee that Linux kernel do not access an unauthorized device. In order to avoid the access to an unauthorized device, the U-boot fixes up the Linux kernel device tree to disable the peripheral nodes which are not assigned to the Cortex-A7 non-secure context.

4.3 Linux kernel

Each assignable peripheral is declared twice in the Linux kernel device tree:

- Once in the **soc** node from arch/arm/boot/dts/stm32mp151.dtsi , corresponding to Linux assigned peripherals
  - Example: i2c2
- Once in the **m4_rproc** node from arch/arm/boot/dts/stm32mp157-m4-srm.dtsi , corresponding to the Cortex-M4 context.

Those nodes are disabled, by default.
  - Example: m4_i2c2

In the board device tree file (*.dts), each assignable peripheral has to be enabled only for the context to which it is assigned, in line with TF-A configuration.
As a consequence, a peripheral assigned to the Cortex-A7 secure has both nodes disabled in the Linux device tree.

Example:

```
&i2c2 {
        status = "disabled";
};
...
&m4_i2c2 {
        status = "okay";
};
```

Information

In addition of this assignment, the user may have to complete the system resources configuration in the device tree nodes. Refer to How to configure system resources for details.

4.4 STM32Cube

There is no configuration to do on STM32Cube side regarding the assignment and isolation. Nevertheless, the resource manager utility, relying on ETZPC configuration, can be used to check that the corresponding peripheral is well assigned to the Cortex-M4 before using it.

Example:

```
int main(void)
{
...
 /* Initialize I2C2------------------------------------------------------ */
 /* Ask the resource manager for the I2C2 resource */
 ResMgr_Init(NULL, NULL);
 if (ResMgr_Request(RESMGR_ID_I2C2, RESMGR_FLAGS_ACCESS_NORMAL | \
```

```
                                                    RESMGR_FLAGS_CPU1, 0, NULL) != RESMGR_OK)
  {
    Error_Handler();
  }
 ...
  if (HAL_I2C_Init(&I2C2) != HAL_OK)
  {
    Error_Handler();
  }
 }
```

4.5 OP-TEE

The OP-TEE OS may use STM32MP1 resources. OP-TEE STM32MP1 drivers register the device driver they intend to used in a secure context. This information is used to consolidate system configuration including secure hardening of configurable peripherals.

In most case, the OP-TEE driver probe relies on OP-TEE device tree porperty *secure-status = "okay"*.

Cortex®

Trusted Firmware for Arm Cortex-A

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

Linux® is a registered trademark of Linus Torvalds.

Inter-Integrated Circuit (Bi-directional 2-wire bus standard for efficient inter-IC control.)

Open Portable Trusted Execution Environment

Hardware Abstraction Layer

Operating System

Microcontroller Unit (MCUs have internal flash memory and are intended to operate with a minimum amount of external support ICs. They commonly are a self-contained, system-on-chip (SoC) designs.)

Extended TrustZone Protection Controller

Stable: 13.05.2020 - 08:56 / Revision: 13.05.2020 - 08:54

A quality version of this page, approved on *13 May 2020*, was based off this revision.

Contents

1 Overview of the OP-TEE open source project ... 18
2 Architecture ... 19
2.1 OP-TEE core ... 19
2.2 OP-TEE trusted libraries ... 19
2.3 TEE Linux driver ... 20
2.4 TEE Client API ... 20
2.5 TEE supplicant ... 20
2.6 Host tools ... 20
3 Booting with OP-TEE ... 21
4 Invoking the OP-TEE services from Linux based OS ... 22
5 Experiencing OP-TEE on a target ... 23

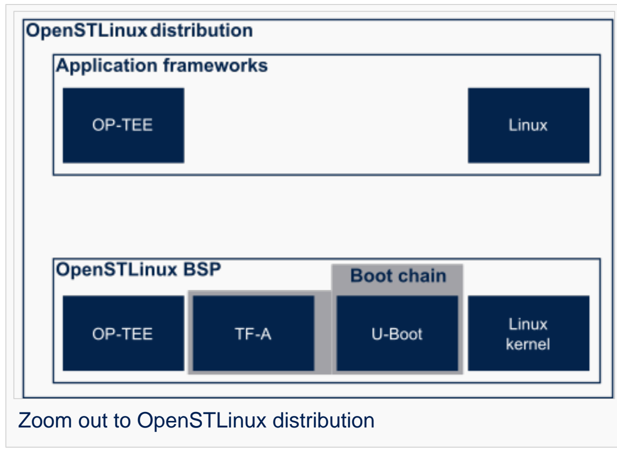
ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

6 References 24

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

1 Overview of the OP-TEE open source project

OP-TEE allows the development and integration of secure services and applications under trusted execution environments, that is execution environments isolated from the Linux® -based OS .



Zoom out to OpenSTLinux distribution

Description extracted from the OP-TEE site[1]:

> *"OP-TEE is an open source project, which contains a full implementation to make up a complete Trusted Execution Environment using the ARM®TrustZone®. technology. OP-TEE meets the GlobalPlatform TEE System Architecture specification. It also provides the TEE Internal core API v1.1 as defined by the GlobalPlatform TEE Standard for the development of Trusted Applications. OP-TEE Trusted OS is accessible from the Linux based OS using the GlobalPlatform TEE Client API Specification v1.0, which also is used to trigger secure execution of applications within the TEE."*

OP-TEE is delivered under a BSD style license and can run secure (trusted) applications without restriction on their licensing model.

The OP-TEE project is maintained by the Linaro Security Working Group.

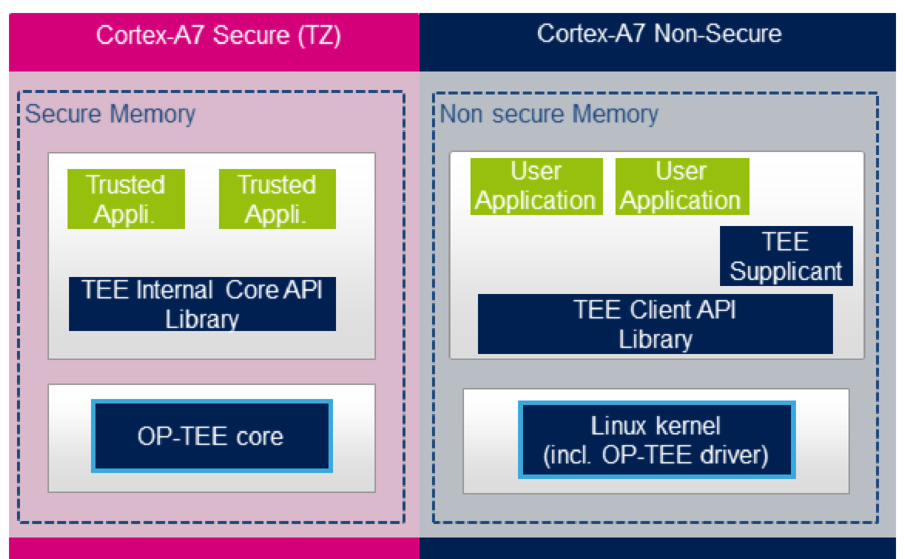
- OP-TEE official site[1]
- OP-TEE source repositories [2][3][4]
- OP-TEE documentation[5]

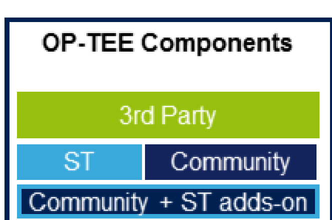
GlobalPlatform Device TEE specifications (TEE Client API, TEE Internal Core API and few more) is available from the GlobalPlatform site[6].

2 Architecture

The OP-TEE project includes several secure and non-secure embedded components, as well as some tools for development and debugging purposes.

The figure below shows the main OP-TEE embedded components, namely the OP-TEE core and trusted application standard libraries on the secure side, and the Client API library, the OP-TEE supplicant daemon and the OP-TEE Linux kernel driver on the non-secure side.





2.1 OP-TEE core

The main OP-TEE component is the OP-TEE core. The OP-TEE core execution is done in Arm®Cortex®-A secure state while the non-secure world (likely a Linux based OS) is done in the non-secure state of the processor. The OP-TEE core executes in secure privileged (kernel) mode, while trusted applications are executed in secure user mode.

OP-TEE can load signed trusted applications stored in the Linux OS file system or embedded in the OP-TEE core boot image.

On devices with secure external memory, the OP-TEE core runs as a monolithic image in the secure memory. On devices with a small secure memory, the OP-TEE core can run in paging-on-demand configuration: a small resident agent is loaded in the small secure memory and can securely page-in/page-out data from/to the non-secure (or less secure) external memory.

OP-TEE core source files can be found from optee_os repository [2].

2.2 OP-TEE trusted libraries

OP-TEE embeds utility libraries for trusted application development including the GlobaPlatform Device TEE Internal Core API Library, which provides the standard services a trusted application can expect from the TEE. OP-TEE supports the loading of static and dynamic libraries in the TEE.

The OP-TEE standard trusted application libraries source files can be found in the optee_os repository[2].

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

2.3 TEE Linux driver

The OP-TEE Linux driver is part of the Linux kernel since release 4.12.

The OP-TEE Linux driver is enabled via the CONFIG_OPTEE configuration directive through the usual Linux kernel configuration means. The driver can be probed thanks to a device tree node.

2.4 TEE Client API

The OP-TEE project embeds an implementation of the GlobaPlatform Device TEE Client API specification for Linux based OS. This TEE Client API specification is partly implemented as a userland library and partly as a Linux kernel OP-TEE driver. The API allows userland clients to invoke trusted applications and the OP-TEE core services exported to non-secure world with a standard API.

The OP-TEE Client API library source files can be found in the optee_client repository[3].

2.5 TEE supplicant

The OP-TEE core can rely on non-secure remote services. OP-TEE embeds an implementation of a non-secure userland supplicant, that can be invoked by the OP-TEE core through the OP-TEE Linux kernel driver. An example of such service is the access to a non-volatile media device that is controlled in the non-secure world.

The OP-TEE supplicant source files can be found in the optee_client repository[3].

2.6 Host tools

The OP-TEE optee_os component, once built, generates a so-called Trusted Application Development Kit to ease the development and integration of trusted applications on a target system. The Trusted Application Development Kit includes the libraries, with their header files and makefile scripts, that allow the generation of signed trusted applications from their respective source files.

Optee_os package also provides a tool to analyse call stack backtraces in case of trusted application and/or OP-TEE core crash. Refer to script **symbolize.py** in optee_os source tree[2].

3 Booting with OP-TEE

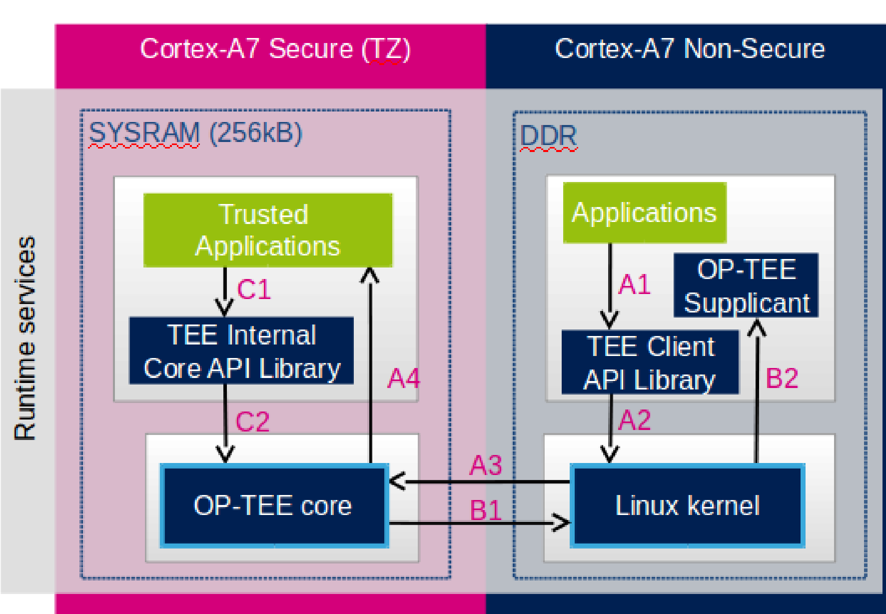
The OP-TEE core is a secure firmware. It must be booted prior to the non-secure world on Arm Cortex-A core(s). The secure bootloader must therefore load the OP-TEE core images in memory and run its initialization prior to executing the first booted non-secure image.

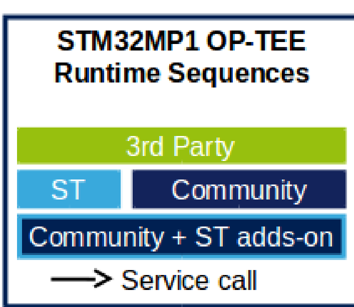
Refer to the target system boot sequences for more details.

4 Invoking the OP-TEE services from Linux based OS

Once the Linux kernel is booted, the OP-TEE core is already initialized and ready to serve.

The figure below shows the main run time sequences in which the OP-TEE can be involved.





Sequence A: an non-secure application invokes a service from a trusted application.

The non-secure application calls the TEE Client API library (**A1**), which in turns invokes (**A2**) the Linux kernel OP-TEE driver. The OP-TEE driver invokes the secure world (**A3**) and reaches the OP-TEE core. The last OP-TEE core transfers the request (**A4**) to the target trusted application. Once the trusted application has completed the request, the system branches back to the calling application with the request status.

If an invoked trusted application is not yet loaded into the TEE, the OP-TEE core loads it by calling remote services through the non-secure TEE supplicant as described in **sequence B** below.

In addition, any invocation of the TEE from the non-secure world must go through the Linux kernel OP-TEE driver.

Sequence B: the OP-TEE core must invoke a non-secure remote service.

The OP-TEE core invokes (**B1**) the Linux kernel OP-TEE driver which in turns notifies the TEE supplicant daemon (**B2**) for a request. Once the supplicant has completed the request, the system branches back to the OP-TEE core with the request status.

Sequence C: a trusted application invokes an OP-TEE core service.

Most of the services defined by the GlobaPlatform Device TEE Internal Core API must be executed in OP-TEE core privileged mode. The trusted application calls the corresponding service from the TEE Internal Core API library (**C1**), which issues a system call (**C2**) to the OP-TEE core. Once the core has completed the request, the system branches back to the calling trusted application with the request status.

5 Experiencing OP-TEE on a target

First make sure your setup includes OP-TEE in the boot sequence. If the OP-TEE core console traces are enabled, you should see the OP-TEE banner after secure bootloader traces and before non-secure bootloader traces. The OP-TEE core banner looks like this:

```
I/TC:  OP-TEE version: <some-reference-version-info> #1 Mon Jun 25 08:59:21 UTC 2018 arm
I/TC:  Initialized
```

The Linux kernel boot traces also show the successful probing of the OP-TEE Linux kernel driver:

```
optee: probing for conduit method from DT.
optee: initialized driver
```

The OP-TEE non-secure components are stored in the file system:

- By default the TEE supplicant is installed at **/usr/bin/tee-supplicant**.
- By default, the TEE Client API library is installed at **/usr/lib/teec.so**.
- By default the TEE regression test tool is installed at **/usr/bin/xtest**.

In the default OP-TEE configuration, trusted applications are stored in the non-secure filesystem at **/lib/optee_armtz/*.ta**.

OP-TEE provides means to protect the trusted application binary images from corruption as image signature or installation in the OP-TEE secure storage. In any case, it is likely that Othe P-TEE core needs to invoke a non-secure service to retrieve the trusted application(s) from some non-secure filesystem data in order to load trusted application(s) in the TEE. This service requires the availability of the OP-TEE supplicant.

Therefore, once the non-secure OS has booted, it must launch the OP-TEE supplicant as a background daemon. Use the following shell command to start the OP-TEE supplicant from a booted Linux system, :

```
sh> tee-supplicant &
```

The OP-TEE package comes with some examples and regression tests. Use the following embedded shell command to run the regression tests:

```
sh> xtest
```

or to run only selective tests:

```
sh> xtest 1002     # Invokes some OP-TEE internal core services
sh> xtest 1004     # Invokes a trusted application loaded from the non-secure filesystem
```

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

6 References

- [1.01.1] https://op-tee.org
- [2.02.12.22.3] https://github.com/OP-TEE/optee_os
- [3.03.13.2] https://github.com/OP-TEE/optee_client
- https://github.com/OP-TEE/optee_test
- https://optee.readthedocs.io/
- https://globalplatform.org/

Open Portable Trusted Execution Environment

Linux® is a registered trademark of Linus Torvalds.

Operating System

TrustZone®

Arm® and TrustZone® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Trusted Execution Environment

Application programming interface

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

Device Tree

Stable: 23.09.2020 - 13:22 / Revision: 12.06.2020 - 13:25

A quality version of this page, approved on *23 September 2020*, was based off this revision.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

1 STM32CubeMX overview

This article describes STM32CubeMX, an official STMicroelectronics graphical software configuration tool.

The STM32CubeMX application helps developers to use the STM32 by means of a user interface, and guides the user through to the initial configuration of a firmware project.

It provides the means to:

- configure pin assignments, the clock tree, or internal peripherals
- simulate the power consumption of the resulting project
- configure and tune DDR parameters
- generate HAL initialization code for Cortex-M4
- generate the Device Tree for a Linux kernel, TF-A and U-Boot firmware for Cortex-A7

It uses a rich library of data from the STM32 microcontroller portfolio.

The application is intended to ease the initial development phase by helping developers to select the best product in terms of features and power.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

2 STM32CubeMX main features

- Peripheral and middleware parameters

  Presents options specific to each supported software component
- Peripheral assignment to processors

  Allows assignment of each peripheral to Cortex-A Secure, Cortex-A Non-Secure, or Cortex-M processors
- Power consumption calculator

  Uses a database of typical values to estimate power consumption, DMIPS, and battery life
- Code generation

  Makes code regeneration possible, while keeping user code intact
- Pinout configuration

  Enables peripherals to be chosen for use, and assigns GPIO and alternate functions to pins
- Clock tree initialization

  Chooses the oscillator and sets the PLL and clock dividers
- DDR tunning tool

  Ensures the configuration, testing, and tuning of the MPU DDR parameters. Using U-Boot-SPL Embedded Software.

3 How to get STM32CubeMX

Please, refer to the following link STM32CubeMX to find STM32CubeMX, the Release Note, the User Manual and the product specification.

Doubledata rate (memory domain)

Hardware Abstraction Layer

Cortex®

Linux® is a registered trademark of Linus Torvalds.

Trusted Firmware for Arm Cortex-A

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)

Microprocessor Unit

Stable: 17.11.2020 - 17:06 / Revision: 10.11.2020 - 07:49

A quality version of this page, approved on *17 November 2020*, was based off this revision.

All the resources for the STM32MP1 Series are located in the Resources area of the STM32MP1 Series web page.

The resources below are referenced in some of the articles of this user guide.

Information

The different **STM32MP15** microprocessor **part numbers** available (with their corresponding internal peripherals, security options and packages) are described in the **STM32MP15 microprocessor part numbers**.

means that the document (or its version) is new compared to what was delivered within the previous ecosystem release.

Reference	Name	Link	Version
Application notes			
AN4803	High-speed SI simulations using IBIS and board-level simulations using HyperLynx® SI on STM32 MCUs and MPUs	AN4803.pdf	v2.0
AN5027	Interfacing PDM digital microphones using STM32 MCUs and MPUs	AN5027.pdf	v2.0
AN5031	Getting started with STM32MP15 Series hardware development	AN5031.pdf	v2.0
		AN503	

Reference	Name	Link	Version
Application notes			
AN5036	Thermal management guidelines for STM32 applications	6.pdf	v3.0
AN5109	STM32MP1 Series using low-power modes	AN5109.pdf	NEW v4.0
AN5122	STM32MP1 Series DDR memory routing guidelines	AN5122.pdf	v3.0
AN5168	STM32MP1 series DDR configuration	AN5168.pdf	v1.0
AN5225	USB Type-C™ Power Delivery using STM32xx Series MCUs and STM32xxx Series MPUs	AN5225.pdf	NEW v3.0
AN5253	Migration of microcontroller applications from STM32F4x9 lines to STM32MP151, STM32MP153 and STM32MP157 lines microprocessor	AN5253.pdf	v1.0
AN5256	STM32MP151, STM32MP153 and STM32MP157 discrete power supply hardware integration	AN5256.pdf	v2.0
AN5260	STM32MP151/153/157 MPU lines and STPMIC1B integration on a battery powered application	AN5260.pdf	v1.0
AN5275	USB DFU/USART protocols used in STM32MP1 Series bootloaders	AN5275.pdf	v1.0
AN5284	STM32MP1 series system power consumption	AN5284.pdf	v1.0
AN5348	FDCAN peripheral on STM32 devices	AN5348.pdf	v1.0
AN5431	The STPMIC1 PCB layout guidlines	AN5431.pdf	v1.0
AN5438	STM32MP1 Series lifetime estimates	AN5438.pdf	v1.0
AN5510	Overview of the secure secret provisioning (SSP) on STM32MP1 Series	AN5510.pdf	v1.0
Datasheets[1]			
DS12505	STM32MP157C/F datasheet (secure)	DS12505.pdf	NEW v4.0
DS12504	STM32MP157A/D datasheet (basic)	DS12504.pdf	NEW v4.0
DS12503	STM32MP153C/F datasheet (secure)	DS12503.pdf	NEW v4.0

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Reference	Name	Link	Version
Application notes			
DS12502	STM32MP153A/D datasheet (basic)	DS12502.pdf	NEW v4.0
DS12501	STM32MP151C/F datasheet (secure)	DS12501.pdf	NEW v4.0
DS12500	STM32MP151A/D datasheet (basic)	DS12500.pdf	NEW v4.0
DS12792	STPMIC1 datasheet	stpmic1.pdf	NEW v5.0
Errata sheets			
ES0438	STM32MP15xx device errata	ES0438.pdf	v5.0
Reference manuals[1]			
RM0436	STM32MP157 reference manual (STM32MP157xxx advanced Arm®-based 32-bit MPUs)	RM0436.pdf	v4.0
RM0442	STM32MP153 reference manual (STM32MP153xxx advanced Arm®-based 32-bit MPUs)	RM0442.pdf	v4.0
RM0441	STM32MP151 reference manual (STM32MP151xxx advanced Arm®-based 32-bit MPUs)	RM0441.pdf	v4.0
Boards schematics			
MB1262 schematics	STM32MP157C-EV1 motherboard schematics MB1262-C01 board schematic (Evaluation board)	MB1262-C01.pdf	v1.0
MB1263 schematics	STM32MP157C-EV1 daughterboard schematics MB1263-C01 board schematic (Evaluation board)	MB1263-C01.pdf	v1.0
NEW MB1263 schematics	STM32MP157F-EV1 daughterboard schematics MB1263-C04 board schematic (Evaluation board)	MB1263-C04.pdf	v4.0
MB1230 schematics	DSI 720p LCD display daughterboard schematics MB1230-C board schematic (Evaluation board)	MB1230-C.pdf	v1.1
MB1379 schematics	Camera daughterboard schematics MB1379-A01 board schematic (Evaluation board)	MB1379-A01.pdf	v1.0
MB1272 schematics	STM32MP157x-DKx motherboard schematics MB1272-DK2-C01 board schematic (Discovery kit)	MB1272-C01.pdf	v1.0

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Reference	Name	Link	Version
Application notes			
MB1407 schematics	STM32MP157x-DKx daughterboard schematics MB1407-LCD-C01 board schematic (Discovery kit)	MB1407-C01.pdf	v1.0
Boards user manuals			
UM2535	STM32MP157x-EV1 evaluation board user manual	UM2535.pdf	v2.0
UM2534	STM32MP157x-DKx discovery board user manual	UM2534.pdf	v1.0
Tools user manuals			
UM2563	STM32CubeIDE installation guide	UM2563.pdf	v1.0
UM2579	Migration guide from System Workbench to STM32CubeIDE	UM2579.pdf	v1.0
UM2553	STM32CubeIDE quick start guide	UM2553.pdf	v1.0
AN5360	Getting started with projects based on the STM32MP1 Series in STM32CubeIDE	AN5360.pdf	v1.0
UM2609	Description of the integrated development environment for STM32 products	UM2609.pdf	v1.0
UM1718	STM32CubeMX user manual	UM1718.pdf	NEW v32.0
UM2237	STM32CubeProgrammer tool user manual	UM2237.pdf	NEW v12.0
UM2238	STM32 Trusted Package Creator tool user manual	UM2238.pdf	NEW v7.0
UM2542	STM32 Series Key Generator tool user manual	UM2542.pdf	v1.0
UM2543	STM32 Series Signing tool user manual	UM2543.pdf	v1.0

- 1.01.1 The part numbers are specified in STM32MP15 microprocessor part numbers

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

Archives

STM32MP15 release	ST documentation
STM32MP15-Ecosystem-v2.0.0	STM32MP15 resources - v2.0.0
STM32MP15-Ecosystem-v1.2.0	STM32MP15 resources - v1.2.0 page for the v1 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.1.0	STM32MP15 resources - v1.1.0 page for the v1 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.0.0	STM32MP15 resources - v1.0.0 page for the v1 ecosystem releases (in archived wiki)

Doubledata rate (memory domain)

USB port or connector

Microprocessor Unit

Device Firmware Upgrade

Universal Synchronous/Asynchronous Receiver/Transmitter

Printed Circuit Board

Secure Secret Provisioning

Secure secrets provisioning

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Display Serial Interface (MIPI® Alliance standard)

Stable: 25.09.2020 - 09:15 / Revision: 25.09.2020 - 09:13

A quality version of this page, approved on *25 September 2020*, was based off this revision.

1 STM32MPU Embedded Software overview

The diagram below shows STM32MPU Embedded Software distribution main components:

- The **OpenSTLinux distribution**, running on the Arm®Cortex®-A, including:
  - The **OpenSTLinux BSP** with:
    - The boot chain based on TF-A and U-Boot.
    - The OP-TEE secure OS running on the Arm®Cortex®-A in secure mode.
    - The **Linux® kernel** running on the Arm®Cortex®-A in non-secure mode.
  - The **application frameworks** are composed of middlewares relying on the BSP and providing API:
    - on the **OP-TEE** side to run **Trusted Applications (TA)** that allow to manipulate secrets (not visible from the Linux and STM32Cube MPU Package)
    - on the **Linux** side to run **Applications** that typically interact with the user via the display, the touchscreen, etc.
- The **STM32Cube MPU Package** is running on the Arm®Cortex®-M: it is based on HAL drivers and middlewares, like other STM32 microcontrollers, completed with coprocessor management.

The figure below is clickable so that the user can directly jump to one of the sub-levels listed above.

STM32MPU Embedded Software architecture overview.

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

2 Open Source Software (OSS) philosophy

The **Open source software** source code is released under a license in which the copyright holder grants users the rights to study, change and distribute the software to anyone and for any purpose[1].
STMicroelectronics maximizes the using of open source software and contributes to those communities. Notice that, due to the software review life cycle, it can take some time before getting all developments accepted in the communities, so STMicroelectronics can also temporarily provide some source code on github[2], until it is merged in the targeted repository.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

3 References

- https://en.wikipedia.org/wiki/Open-source_software
- STM32MP1 Distribution Package

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Cortex®

Board support package

Operating System

Linux® is a registered trademark of Linus Torvalds.

Application programming interface

Open Portable Trusted Execution Environment

Trusted Application

Microprocessor Unit

Hardware Abstraction Layer

Open Source Software

Stable: 17.02.2021 - 19:40 / Revision: 16.02.2021 - 16:25

A quality version of this page, approved on *17 February 2021*, was based off this revision.

Contents

1 Trusted Firmware-A 36
2 Architecture 37
3 Boot loader stages 39
3.1 BL1 39
3.2 BL2 39
3.3 BL32 39
4 References 40

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. Arm is transferring the Trusted Firmware project to be managed as an open-source project by Linaro.[1]

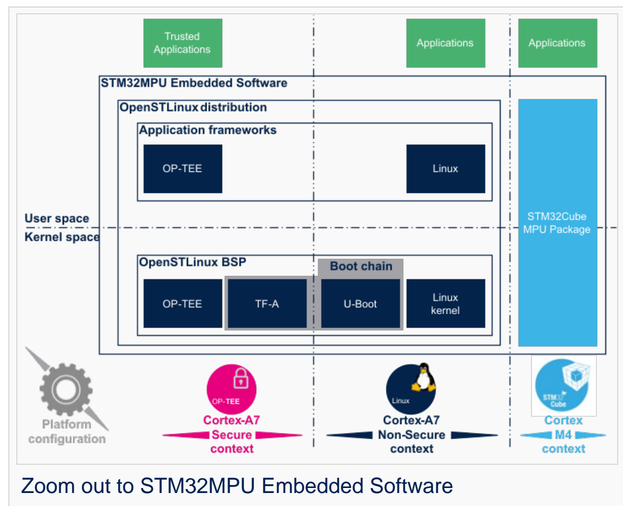
It is used as the first-stage boot loader (FSBL) on STM32 MPU platforms when using the trusted boot chain.

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

Trusted Firmware-A also implements a secure monitor with various Arm interface standards:

- The power state coordination interface (PSCI) [4]
- Trusted board boot requirements (TBBR) [5]
- SMC calling convention [6]
- System control and management interface [7]

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



Zoom out to STM32MPU Embedded Software

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

2 Architecture

The global architecture of TF-A is explained in the Trusted Firmware-A design [8] document.

TF-A is divided into several binaries, each with a dedicated main role. For 32-bit Arm processors (AArch32), it is divided into four steps (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.

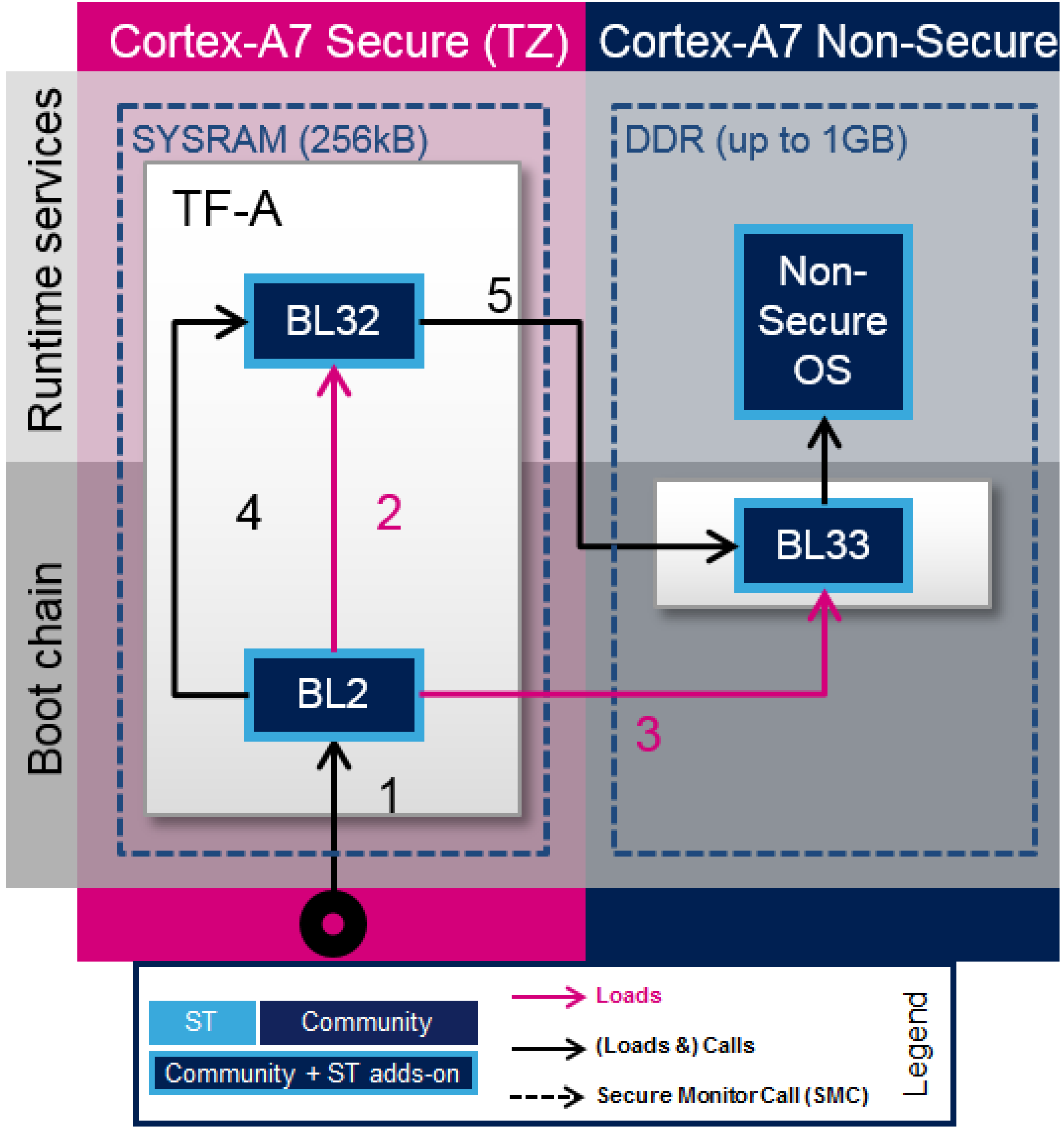
BL1 is now optional, and can be removed by enabling the compilation flag: BL2_AT_EL3. It is then removed for the STM32MP1, as all BL1 tasks are done by ROM code, or BL2.

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, as this is the case on STM32MP1. It is a reduced version of the Linux kernel one, with only the devices used during boot. It can be configured with STM32CubeMX.

In STMicroelectronics' implementation, the 2 binaries, BL2 and BL32, and the device tree are put together in a single binary, to be loaded at once to the SYSRAM by the ROM code.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy



TF-A loading steps:

1. ROM code loads TF-A binary and calls BL2
2. BL2 prepares BL32
3. BL2 loads BL33
4. BL2 calls BL32
5. BL32 calls BL33

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

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 STM32MP1. As the STM32MP1 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 (trusted boot firmware) 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.

It has to initialize the security components.
For the STM32MP15, these security peripherals are:

- boot and security, and OTP control (BSEC internal peripheral)
- extended TrustZone protection controller (ETZPC internal peripheral)
- TrustZone address space controller for DDR (TZC internal peripheral)

BL2 is also in charge of initializing the DDR and clock tree.

The boot peripheral has to be initialized.
On the STM32MP15, it can be one of the following:

- SD-card via the SDMMC internal peripheral
- eMMC via the SDMMC internal peripheral
- NAND via the FMC internal peripheral
- NOR via the QUADSPI internal peripheral

USB (OTG internal peripheral) or UART(USART internal peripheral) are used when Flashing, see STM32CubeProgrammer for more details.

BL2 also integrates image verification and authentication. Authentication is achieved by calling BootROM verification services.

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

3.3 BL32

BL32 provides runtime secure services. In TF-A, the BL32 default implementation is SP-MIN solution. 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. Both solutions (SP-MIN or OP-TEE) are supported by STMicroelectronics for STM32MP1.

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 [6].

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

It also provides STMicroelectronics dedicated services, to access secure peripherals. On the STM32MP1, these services are used to access RCC internal peripheral, PWR internal peripheral, RTC internal peripheral or BSEC internal peripheral.

4 References

- https://www.trustedfirmware.org/
- https://github.com/ARM-software/arm-trusted-firmware
- [3.03.1] readme.rst
- [4.04.14.2] http://infocenter.arm.com/help/topic/com.arm.doc.den0022d/Power_State_Coordination_Interface_PDD_v1_1_DEN0022D.pdf
- Arm DEN0006C-1
- [6.06.1] http://infocenter.arm.com/help/topic/com.arm.doc.den0028b/ARM_DEN0028B_SMC_Calling_Convention.pdf
- http://infocenter.arm.com/help/topic/com.arm.doc.den0056a/DEN0056A_System_Control_and_Management_Interface.pdf
- https://trustedfirmware-a.readthedocs.io/en/latest/design/index.html

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere. arm

First Stage Boot Loader

Microprocessor Unit

Power State Coordination Interface

Secure Monitor Call

Trusted Firmware for Arm Cortex-A

Boot Loader stage 1

Read Only Memory

Boot Loader stage 2

Boot Loader stage 3-2

Boot Loader stage 3-3

Second Stage Boot Loader

Linux® is a registered trademark of Linus Torvalds.

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

One Time Programmed

TrustZone®
Arm® and TrustZone® are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Doubledata rate (memory domain)

Secure digital

former spelling for e•MMC ('e' in italic)

Universal Asynchronous Receiver/Transmitter

Secure Payload minimal

Operating System

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Trusted Execution Environment

Open Portable Trusted Execution Environment