Category:STM32 MPU microprocessor devices

Contents

Stable: 17.06.2020 - 15:27 / Revision: 12.12.2019 - 10:04

Stable: 17:06.2020 - 10:27 | Revision: 12.12.2019 - 10:04

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

Contents

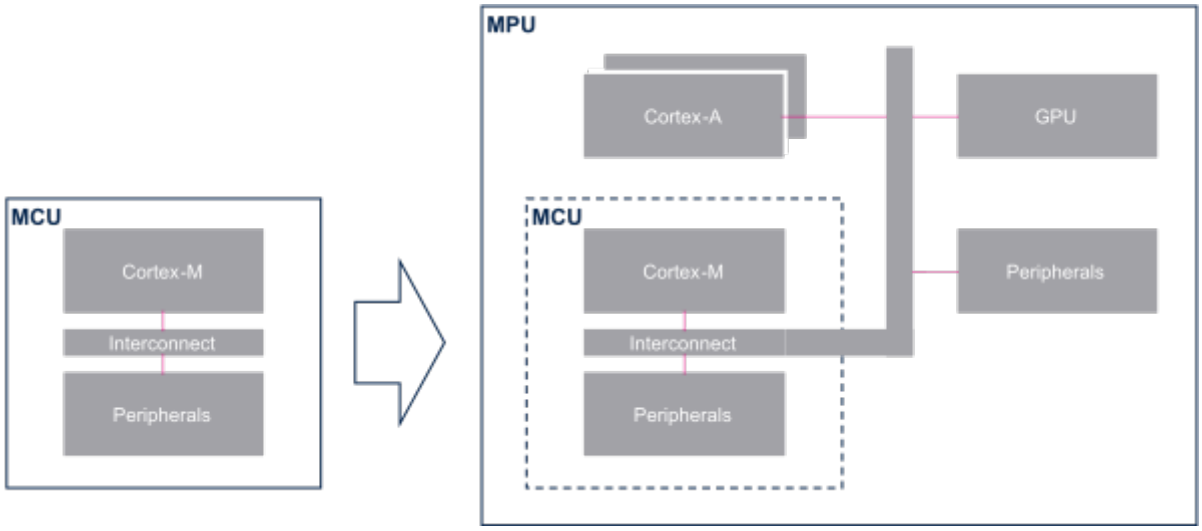
1 Extending the STM32 MCU family to the MPU world

Microcontroller units (MCUs) are built around MMU-less cores such as the Arm Cortex-M, which are very efficient for deterministic operations in a bare metal or real time operating system (RTOS) context. STMicroelectronics STM32 MCUs embed enough SRAM (static RAM) and Flash memory for many applications, and this can be completed with external memories.

Microprocessor units (MPUs) rely on cores such as the Arm Cortex-A, with memory management unit (MMU) to manage virtual memory spaces, opening the door to efficient support of a rich operating system (OS) such as Linux. A fast interconnect makes the bridge between the processing unit, high-bandwidth peripherals, external memories (RAM and NVM) and, usually, a graphical Processing Unit (GPU).

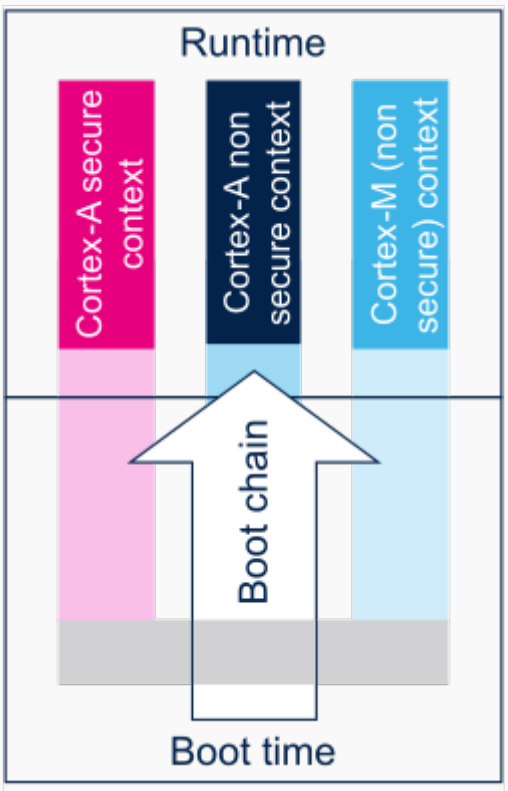
STMicroelectronics has a strong presence in MCU markets with STM32 family [1] and entered the MPU market with a first platform referenced as **STM32MP15**. This platform aims to address multiple market segments such as industrial, consumer, healthcare, home and building automation.

The STMicroelectronics approach for a smooth transition to the MPU world consists of putting both worlds in a single device, seeing the MCU as a subsystem of the MPU:



2 Multiple-core architecture concepts

As seen above, the MPU is a multiple-core architecture that can interact with a wide number of peripherals. Some **new concepts** need to be introduced for a good understanding of the system: these concepts are explained below and are illustrated in the figure on the right.



Runtime context initialization at boot time

2.1 Hardware execution contexts

Each core can run in a non-secure and - eventually - a secure (Arm Trustzone[2]) mode.
A **hardware execution context** corresponds to a core and security mode.
The three hardware execution contexts available on STM32 MPU devices are:

- Arm Cortex-A secure (Trustzone)
- Arm Cortex-A non secure
- Arm Cortex-M (non-secure)

Each hardware execution context can host different firmware, depending on the platform state. The following contexts can be distinguished:

- the **boot time** context, corresponding to a transitory firmware execution, when the device is booting up
- the **runtime** context, corresponding to an established firmware execution, when the device is up-and-running

2.2 Firmwares executed in the runtime contexts

Each runtime context executes a given piece of **firmware**:

- Arm Cortex-A secure (Trustzone), executes **OP-TEE**[3]
- Arm Cortex-A non secure, executes **Linux**[3]

- Arm Cortex-M (non-secure), executes **STM32Cube**[3]

OP-TEE, Linux and STM32Cube are **STM32MPU Embedded Software**[3] components.

2.3 Peripheral assignment to the runtime contexts

The term **peripheral assignment** is used to identify the action to assign a set of peripherals to a runtime context. This is a user choice that can be realized via STM32CubeMX[4] or manually, in order to properly configure the boot chain[5] and the several pieces of firmware that run on the platform.

Each microprocessor peripheral-overview article shows the assignment capabilities for each peripheral, with a table similar to the example below:

Domain Instance	Peripheral	Runtime allocation				Comment
		Cortex-A NS (Linux)	Cortex-M (STM32Cube)			
	YYY1					YYY1 can be assigned (single choice) to whether Cortex-A non-secure or Cortex-M
						YYY2 can only

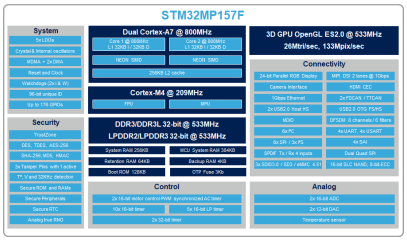
Domain	Peripheral	Runtime allocation				Comment
		YYY2				be assigned to Cortex-A secure
		YYY3				YYY3 is shared accross all contexts: this is typically the case for system peripherals

Refer to How to assign an internal peripheral to a runtime context for detailed instructions.

3 STM32MP1 family microprocessors

What are the main features of an STM32 microprocessor device?
How to program STM32 microprocessor device-internal peripherals?
How to configure internal peripherals for new boards?

Click on the links in the frame below and be guided!



STM32MP15 microprocessor

4 References

- http://www.st.com/en/microcontrollers/stm32-32-bit-arm-cortex-mcus.html
- https://www.arm.com/products/security-on-arm/trustzone
- 3.03.13.23.3 STM32MPU Embedded Software
- STM32CubeMX
- Boot_chain_overview

Subcategories

This category has only the following subcategory.

- Peripherals - Hardware blocks (13 C)

Pages in category "STM32 MPU microprocessor devices"

This category contains only the following page.

- STM32MP15 microprocessor

Stable: 12.03.2021 - 11:29 / Revision: 12.03.2021 - 11:15

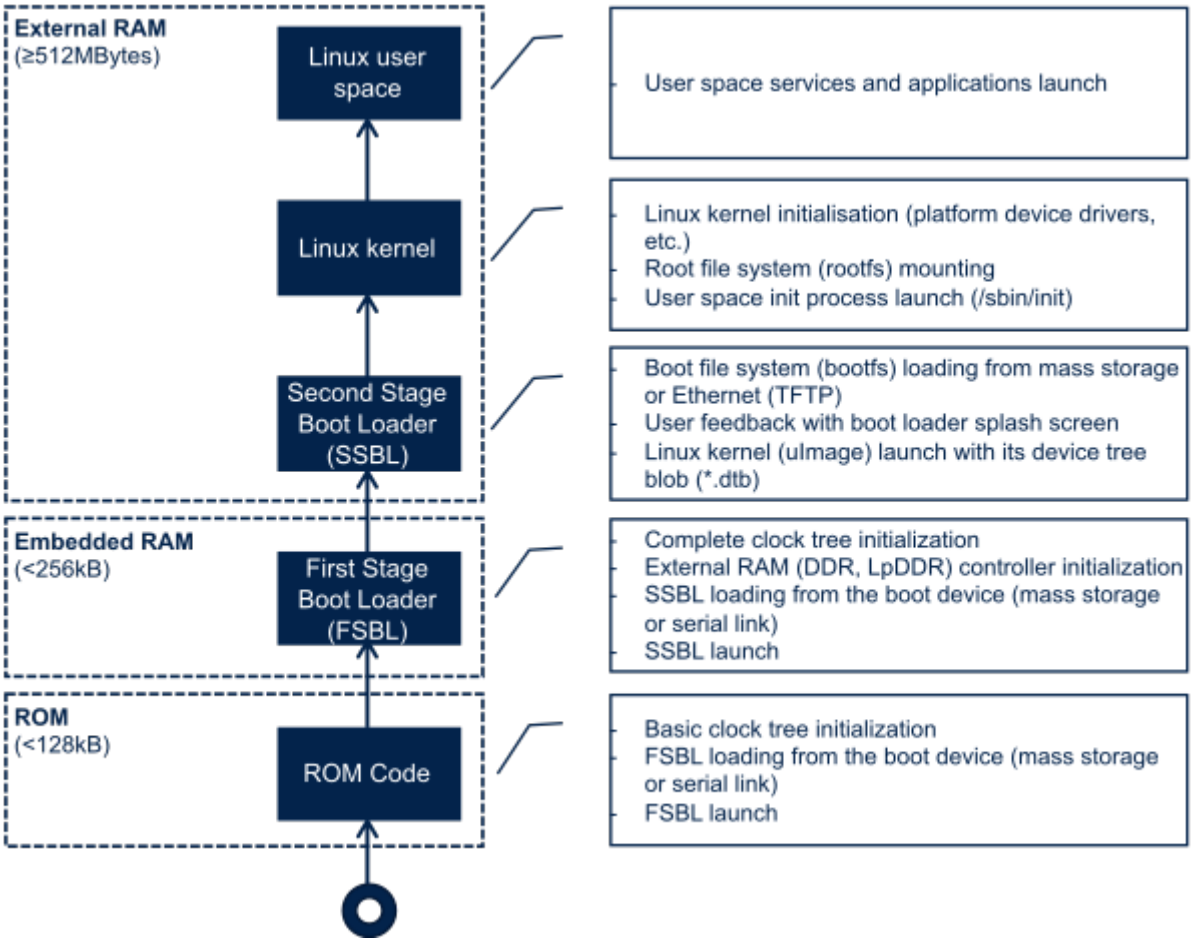
A quality version of this page, approved on *12 March 2021*, was based off this revision.

Contents

1 Generic boot sequence

1.1 Linux start-up

Starting Linux® on a processor is done in several steps that progressively initialize the platform peripherals and memories. These steps are explained in the following paragraphs and illustrated by the diagram on the right, which also gives typical memory sizes for each stage.



1.1.1 ROM code

The ROM code is a piece of software that takes its name from the read only memory (ROM) where it is stored. It fits in a few tens of Kbytes and maps its data in embedded RAM. It is the first code executed by the processor, and it embeds all the logic needed to select the boot device (serial link or Flash) from which the first-stage boot loader (FSBL) is loaded to the embedded RAM.

Most products require to trust the application that is running on the device and the ROM code is the first link in the chain of trust that must be established across all started components: this trust is established by authenticating the FSBL before starting it. In turn, the FSBL and each following component will authenticate the next one, up to a level defined by the product manufacturer.

1.1.2 First stage boot loader (FSBL)

Among other things, the first stage boot loader (FSBL) initializes (part of) the clock tree and the external RAM controller. Finally, the FSBL loads the second-stage boot loader (SSBL) into the external RAM and jumps to it.

The Trusted Firmware-A (TF-A) and U-Boot secondary program loader (U-Boot SPL) are two possible FSBLs.

1.1.3 Second-stage boot loader (SSBL)

The second-stage boot loader (SSBL) runs in a wide RAM so it can implement complex features (USB, Ethernet, display, and so on), that are very useful to make Linux kernel loading more flexible (from a Flash device, a network, and so on), and user-friendly (by showing a splash screen to the user). U-Boot is commonly used as a Linux bootloader in embedded systems.

1.1.4 Linux kernel space

The Linux kernel is started in the external memory and it initializes all the peripheral drivers that are needed on the platform.

1.1.5 Linux user space

Finally, the Linux kernel hands control to the user space starting the init process that runs all initialization actions described in the root file system (rootfs), including the application framework that exposes the user interface (UI) to the user.

1.2 Other services start-up

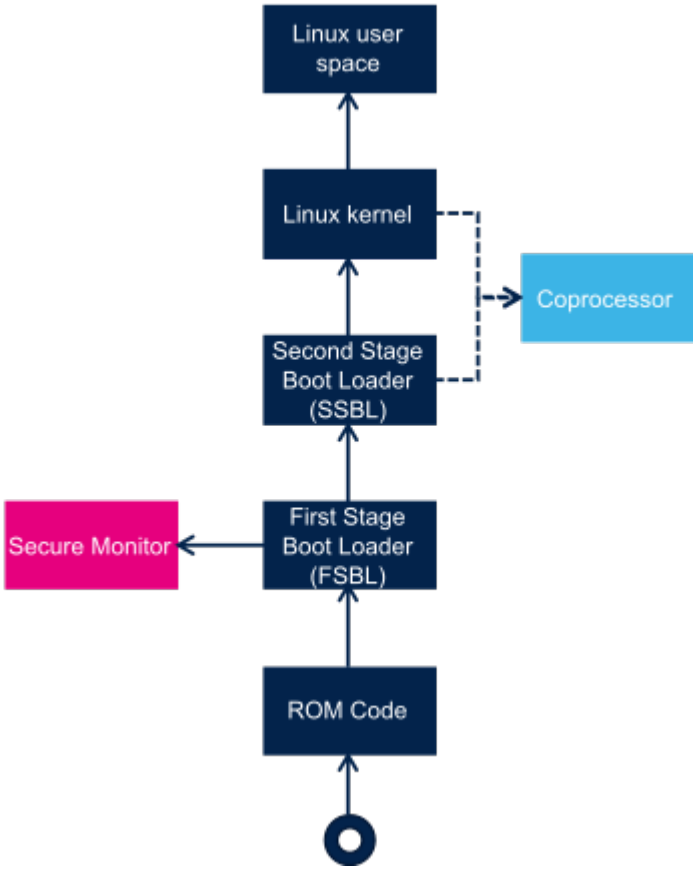
In addition to Linux startup, the boot chain also installs the secure monitor and may support coprocessor firmware loading.

For instance, for the STM32MP15, the boot chain starts:

- the secure monitor, supported by the Arm® Cortex®-A secure context (TrustZone). Examples of use of a secure monitor are: user authentication, key storage, and tampering management.
- the coprocessor firmware, running on the Arm Cortex-M core. This can be used to offload real-time or low-power services.

The dotted lines in the diagram on the right mean that:

- the coprocessor can be started by the **second stage boot loader (SSBL)**, known as “**early boot**”, or **Linux kernel**



2 STM32MP boot sequence

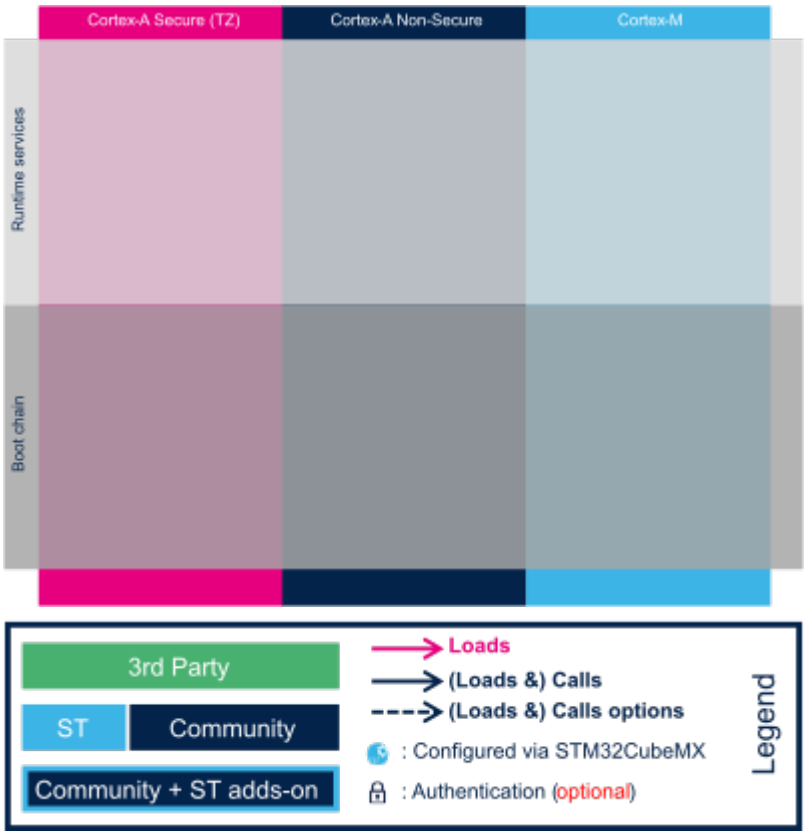
2.1 Diagram frames and legend

The hardware execution contexts are shown with vertical frames in the boot diagrams:

- the Arm Cortex-A secure context, in pink
- the Arm Cortex-A non-secure context, in dark blue
- the Arm Cortex-M context, in light blue

The horizontal frame in:

- the bottom part shows the **boot chain**
- the top part shows the **runtime** services, that are installed by the **boot chain**



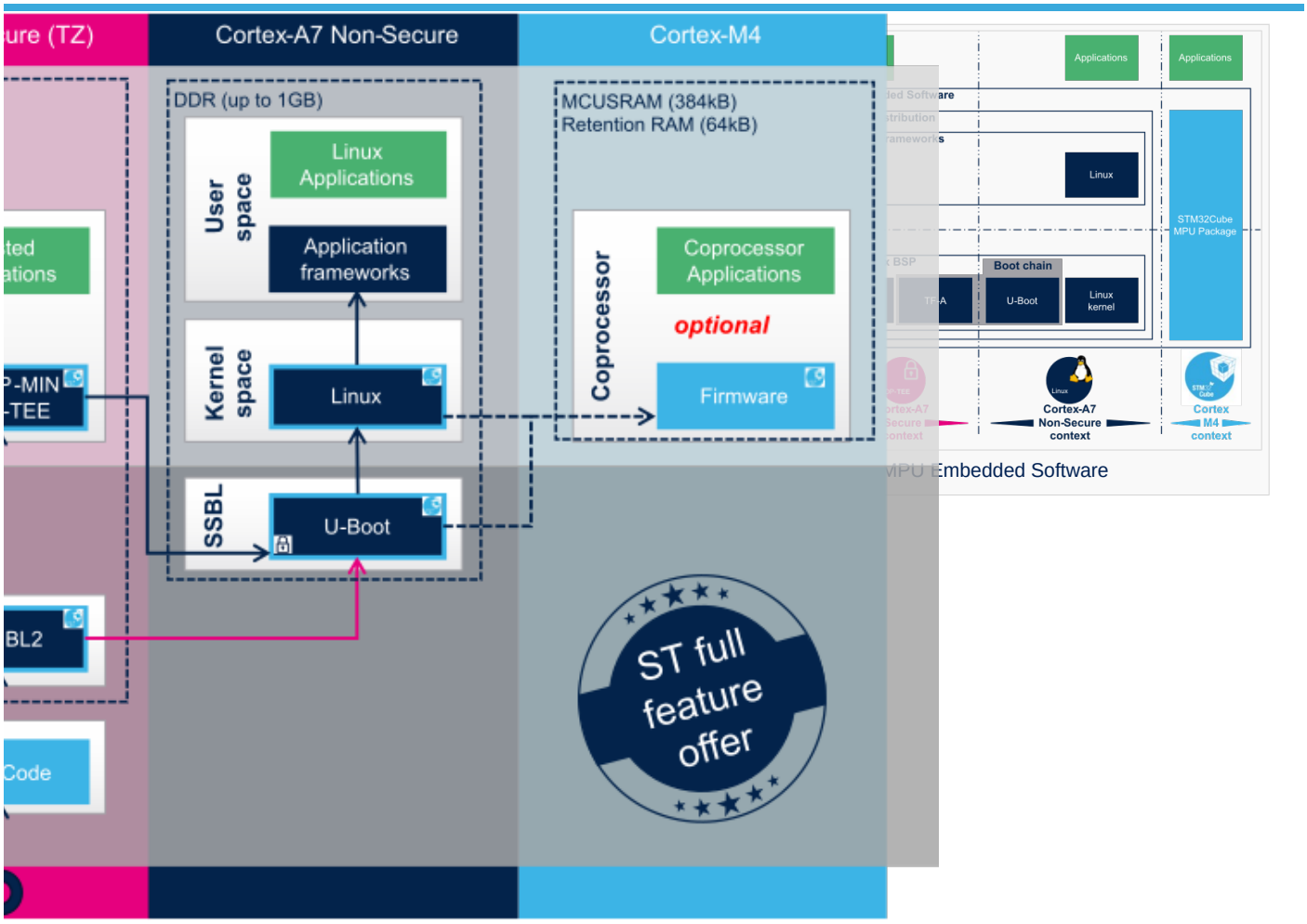
The legend on the right shows how information about the various components shown in the frames, and which are involved in the boot process, is highlighted:

- The box **color** shows the component source code origin
- The **arrows** show the loading and calling actions between the components
- The **Cube** logo is used on the top right corner of components that can be configured via STM32CubeMX
- The **lock** show the components that can be authenticated during the boot process

2.2 STM32MP15 boot chain

2.2.1 Overview

STM32MP15 boot chain uses Trusted Firmware-A (TF-A) as the FSBL in order to fulfill all the requirements for security-sensitive customers, and it uses U-Boot as the SSBL. Note that the authentication is optional with this boot chain, so it can run on any STM32MP15 device security variant (that is, with or without the Secure boot).
Refer to the security overview for an introduction of the secure features available on STM32MP15, from the secure boot up to trusted applications execution.



Note:

- The STM32MP15 coprocessor can be started at the SSBL level by the U-Boot early boot feature or, later, by the Linux remoteproc framework, depending on the application startup time-targets.

2.2.2 ROM code

The ROM code starts the processor in secure mode. It supports the FSBL authentication and offers authentication services to the FSBL.

2.2.3 First stage boot loader (FSBL)

The FSBL is executed from the SYSRAM.
Among other things, this boot loader initializes (part of) the clock tree and the DDR controller. Finally, the FSBL loads the second-stage boot loader (SSBL) into the DDR external RAM and jumps to it.
The boot loader stage 2, so called TF-A BL2, is the Trusted Firmware-A (TF-A) binary used as FSBL on STM32MP15.

2.2.4 Second stage boot loader (SSBL)

U-Boot is commonly used as a bootloader in embedded software and it is the one used on STM32MP15.

2.2.5 Linux

Linux® OS is loaded in DDR by U-Boot and executed in the non-secure context.

2.2.6 Secure OS / Secure Monitor

The Cortex-A7 secure world can implement a minimal secure monitor (from TF-A SP-MIN or U-Boot) or a real secure OS, such as OP-TEE.

2.2.7 Coprocessor firmware

The coprocessor STM32Cube firmware can be started at the SSBL level by U-Boot with the remoteproc feature (rproc command) or, later, by Linux remoteproc framework, depending on the application startup time-targets.

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

User Interface

Stable: 08.03.2021 - 16:13 / Revision: 16.02.2021 - 17:11

A quality version of this page, approved on *8 March 2021*, was based off this revision.

Contents

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.

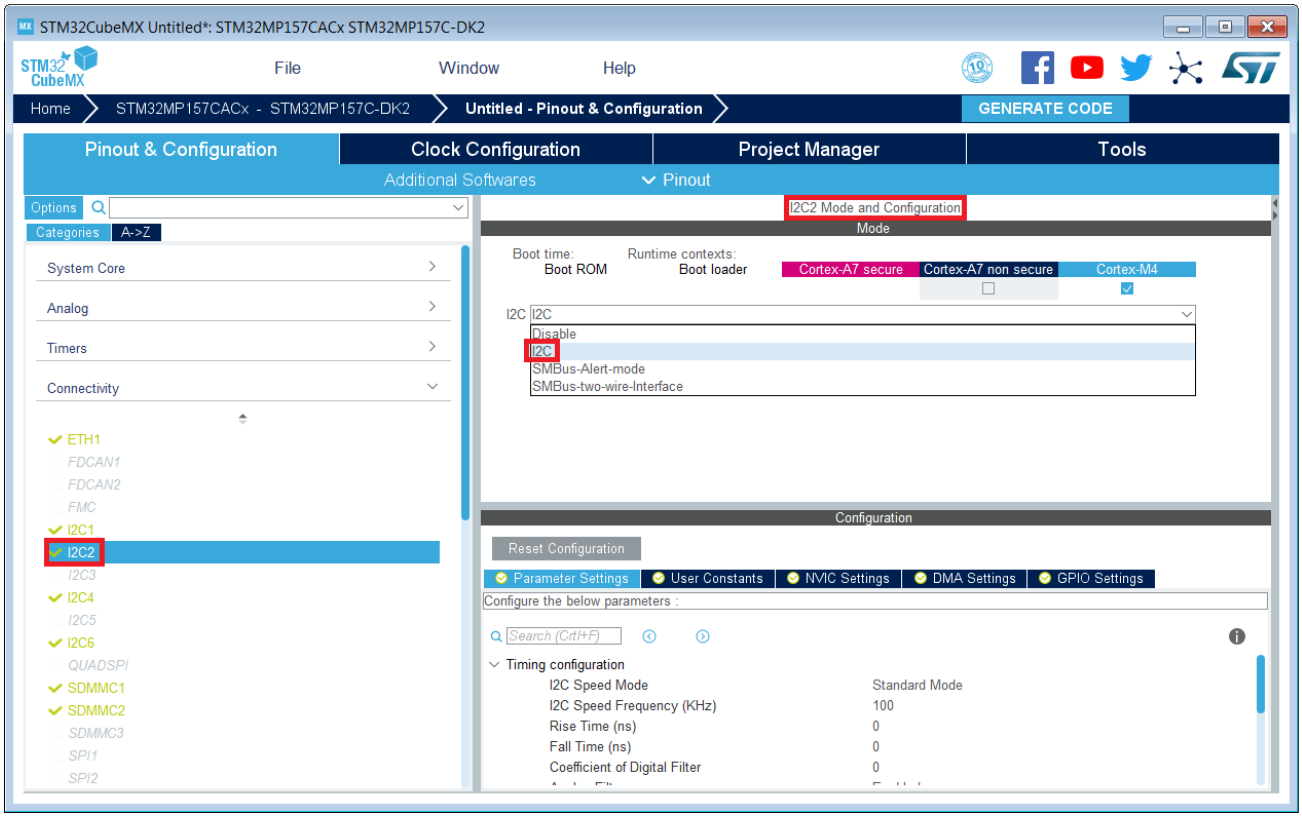
Information

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

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



Information

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.

Information

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.

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/stm32mp15-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"*.

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

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.

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.

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.

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

Stable: 13.10.2021 - 14:27 / Revision: 13.10.2021 - 14:25

A quality version of this page, approved on *13 October 2021*, was based off this revision.

In a first part, this article shows the STM32MP157 line **part number codification** and **block diagram**. STM32MP157 belongs to STM32MP1 Series (refer to the list of part numbers provided below).

The second part of this article digs into technical aspects, and provides entry points to:

- STM32MP15 **documentation**
- articles dedicated to **Internal peripherals** that make the transition towards the software frameworks required to control these peripherals
- the list of **boards** supporting STM32MP15 devices
- the supported **software distributions**, that can be downloaded into the STM32MP15 device.

Contents

1 Introduction

STM32MP15 microprocessors are based on the Arm®**Cortex®-A7** dual core. They support Trustzone mode for secure operations, a **Vivante GPU** and an Arm®**Cortex®-M4 coprocessor**.

Arm® Cortex®-M4 coprocessor and its peripheral set are directly inherited from the STM32 MCU family [1].

2 Part number codification

The table below shows the STM32MP15 microprocessor different part numbers available, together with their corresponding internal peripherals, security options and packages.

2.1 STM32MP15x lines

	Cortex-A7	Cortex-M4	GPU	Display	CAN
STM32MP15**1**	Single	Yes	No	TFT	No
STM32MP15**3**	Dual	Yes	No	TFT	Yes
STM32MP15**7**	Dual	Yes	Yes	TFT/DSI	Yes

2.2 Security and Cortex-A7 frequency

	Security	Cortex-A7 frequency
STM32MP15x**A**	Basic	650 MHz[2]
STM32MP15x**C**	Secure boot + Cryptography (CRYP)	650 MHz[2]
STM32MP15x**D**	Basic	800 MHz[2][3]
STM32MP15x**F**	Secure boot + Cryptography (CRYP)	800 MHz[2][3]

2.3 Packages

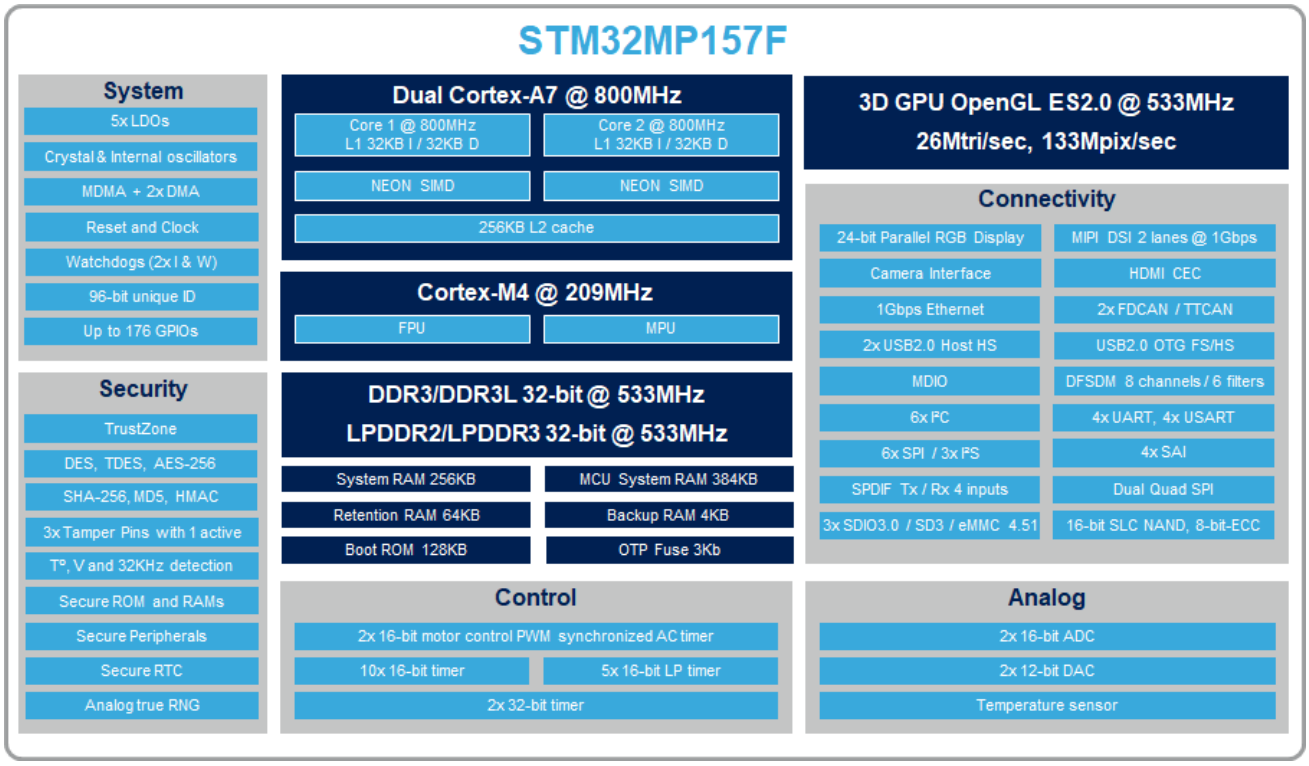
STM32MP15xx**AA**	TFBGA448 18x18
STM32MP15xx**AB**	LFBGA354 16x16
STM32MP15xx**AC**	TFBGA361 12x12
STM32MP15xx**AD**	TFBGA257 10x10

2.4 Junction temperature

STM32MP15xxxx**1**	- 20 to + 105 °C[2][3]
STM32MP15xxxx**3**	- 40 to + 125 °C[2]

3 Block diagram

Here below is the STM32MP157F block diagram offering the richest features set of the STM32MP15 microprocessor.



Notice that the diagram above is a functional view that is not fully aligned with the real design. For instance, SPDIF RX and SPDIF TX functions are grouped in a single box whereas SPDIF RX is implemented in one dedicated peripheral and SPDIF TX is supported by SAI.

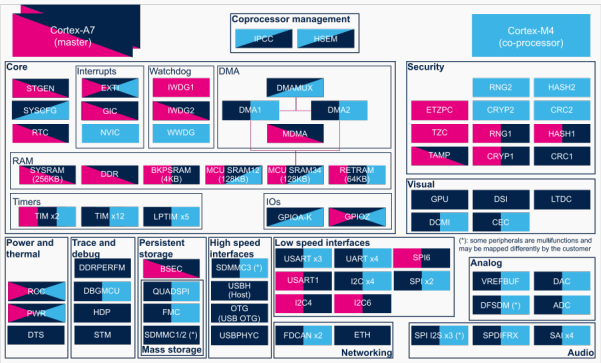
4 Technical documentation

- STM32MP15 Reference Manual: device and internal peripheral user specifications
- STM32MP15 Datasheet: electrical characteristics, package and pinout descriptions

5 Internal peripherals

STM32MP15 peripherals overview article gives a description of all the internal peripherals available on STM32MP15 devices, with direct links to the articles where you can find:

- an overview of each peripheral
- the list of instances available for each peripheral type,
- information on the way each instance can be shared between Arm® Cortex®-A7 and Cortex®-M4 cores,
- direct links to the software frameworks used to control the peripheral from different Arm® cores and security modes such as Cortex®-A7 non secure, Cortex®-A7 secure or Cortex®-M4 (non secure).



Internal peripherals

6 How to get further with STM32MP15 ecosystem

6.1 Boards

The list of boards that integrate STM32MP15 devices can be found in STM32MP15 boards article.

6.2 Supported software distributions

STM32MPU Embedded Software distribution	**STM32MPU Embedded Software distribution for Android**

Click the links above to find information on:

- Distribution composition and associated software architecture
- Associated release notes

7 References and foot notes

- STM32 MCU family
- 2.02.12.22.32.42.5 Exposure to maximum rating conditions for extended periods may affect device reliability. Device mission profile (application conditions) is compliant with JEDEC JESD47 qualification standard. Refer to the STM32MP15 Datasheet and AN5438 for further information.
- 3.03.13.2 800 MHz part numbers are only available with '1' as junction temperatures range suffix (- 20 to + 105 °C).

Stable: 26.03.2021 - 11:32 / Revision: 12.03.2021 - 11:07

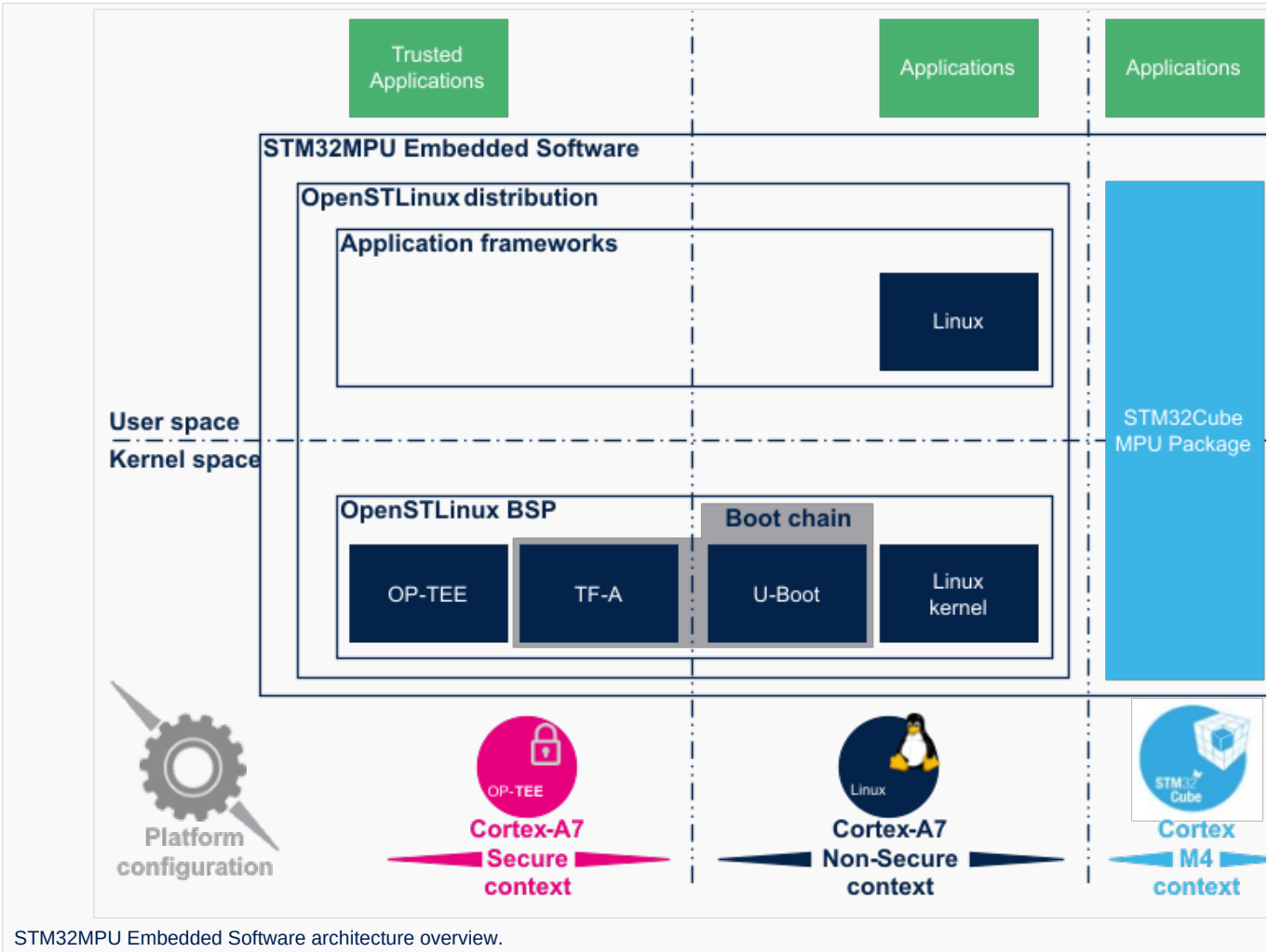
A quality version of this page, approved on *26 March 2021*, was based off this revision.

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 **Linux** side, to run **Applications** that typically interact with the user via the display, the touchscreen, etc.
  - On **OP-TEE** side, the **Trusted Applications (TA)** relies on the OP-TEE core for secrets operations (not visible from the Linux and STM32Cube MPU Package)
- 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.


3rd Party
ST
Community
Legend

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.

3 References

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