



## FDCAN internal peripheral



---

## Contents

---

1. FDCAN internal peripheral .....	3
2. CAN overview .....	9
3. FDCAN device tree configuration .....	18
4. How to assign an internal peripheral to a runtime context .....	24
5. STM32CubeMP1 architecture .....	31
6. STM32CubeMX .....	43
7. STM32MP15 resources .....	46
8. STM32MPU Embedded Software architecture overview .....	50



A quality version of this page, approved on 26 September 2019, 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



---

## 1 Article purpose

---

The purpose of this article is to:

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



---

## 2 Peripheral overview

---

**FDCAN** peripheral handles data communication in a Controller Area Network (CAN) bus system using message-based protocol originally designed for in-vehicle communication. The CAN subsystem consists of two CAN modules (FDCAN1 and FDCAN2), a shared message RAM and an optional clock calibration unit.

### 2.1 Features

Both FDCAN instances are compliant with classic CAN protocol<sup>[1]</sup> and CAN FD<sup>[2]</sup> (CAN with Flexible Data-Rate) protocol. In addition, FDCAN1 supports time triggered CAN (TTCAN).

FDCAN1 and FDCAN2 share a dedicated 10 Kbyte CAN SRAM for message transfers.

Refer to [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

FDCAN is a **non secure** peripheral.



### 3 Peripheral usage and associated software

#### 3.1 Boot time

The FDCAN is not used at boot time.

#### 3.2 Runtime

##### 3.2.1 Overview

FDCAN instances can be allocated to:

- the Arm® Cortex®-A7 non-secure core to be controlled in Linux® by the NetDev framework (See CAN overview)
- or
- the Arm® Cortex®-M4 to be controlled in STM32Cube MPU Package by STM32Cube FDCAN driver

##### 3.2.2 Software frameworks

Domain	Peripheral	Software components		Comment
OP-TEE	Linux	STM32Cube		
Networking	FDCAN		Linux net/can framework	STM32Cube FDCAN driver

##### 3.2.3 Peripheral configuration

The configuration is applied by the firmware running in the context to which the peripheral is assigned. The configuration can be done alone via the STM32CubeMX tool for all internal peripherals, and then manually completed (particularly for external peripherals) according to the information given in the corresponding software framework article. When the FDCAN peripheral is assigned to the Linux® OS, it is configured through the device tree according to the information given in the FDCAN device tree configuration article.

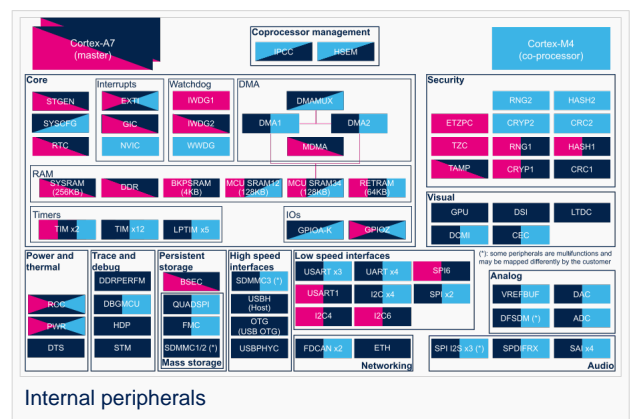
##### 3.2.4 Peripheral assignment

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 possibilities might be described in STM32MP15 reference manuals





Domain	Periphera	Runtime allocation				Comment
Instance	Cortex-A7 secure (OP-TEE)	Cortex-A7 non-secure (Linux)	Cortex-M4  (STM32Cube)			
Networking	FDCAN	FDCAN1				Assignment (single choice)
		FDCAN2				Assignment (single choice)



---

## 4 How to go further

---

### Information

Use this paragraph to add more information and introduce other documentation such as Application Notes (AN)





## 5 References

- CAN protocol implementations, from the CAN in Automation group (CiA)
- CAN FD - The basic idea, from the CAN in Automation group (CiA)

Stable: 01.12.2020 - 10:52 / Revision: 10.06.2020 - 14:46

A quality version of this page, approved on 1 December 2020, was based off this revision.

This article gives information about the Linux<sup>®</sup> Controller Area Network (CAN) framework. It explains how to activate the CAN interface and, based on examples, how to use it.

### Contents

1 Framework purpose .....	10
2 System overview .....	11
2.1 Component description .....	11
2.2 API description .....	12
3 Configuration .....	13
3.1 Kernel configuration .....	13
3.2 Device tree configuration .....	13
4 How to use the framework .....	14
4.1 How to set up a SocketCAN interface .....	14
4.2 How to send/receive CAN data .....	14
5 How to trace and debug the framework .....	15
5.1 How to trace .....	15
5.2 How to monitor CAN bus .....	15
6 Source code location .....	16
7 To go further .....	17
8 References .....	18



---

## 1 Framework purpose

---

The **Controller Area Network** (CAN) is a multi-master serial bus standard connecting at least two nodes. It is a message-based protocol originally designed for in-vehicle communication and which main benefits are a significant reduction of wiring and the prevention of message collision.

For better real-time performance, CAN with Flexible Data-Rate (CAN FD)<sup>[1]</sup> is used as an extension to the classic CAN protocol<sup>[2]</sup>. It allows data rates higher than 1 MBit/s and payloads longer than 8 bytes per frame (up to 64 data bytes).

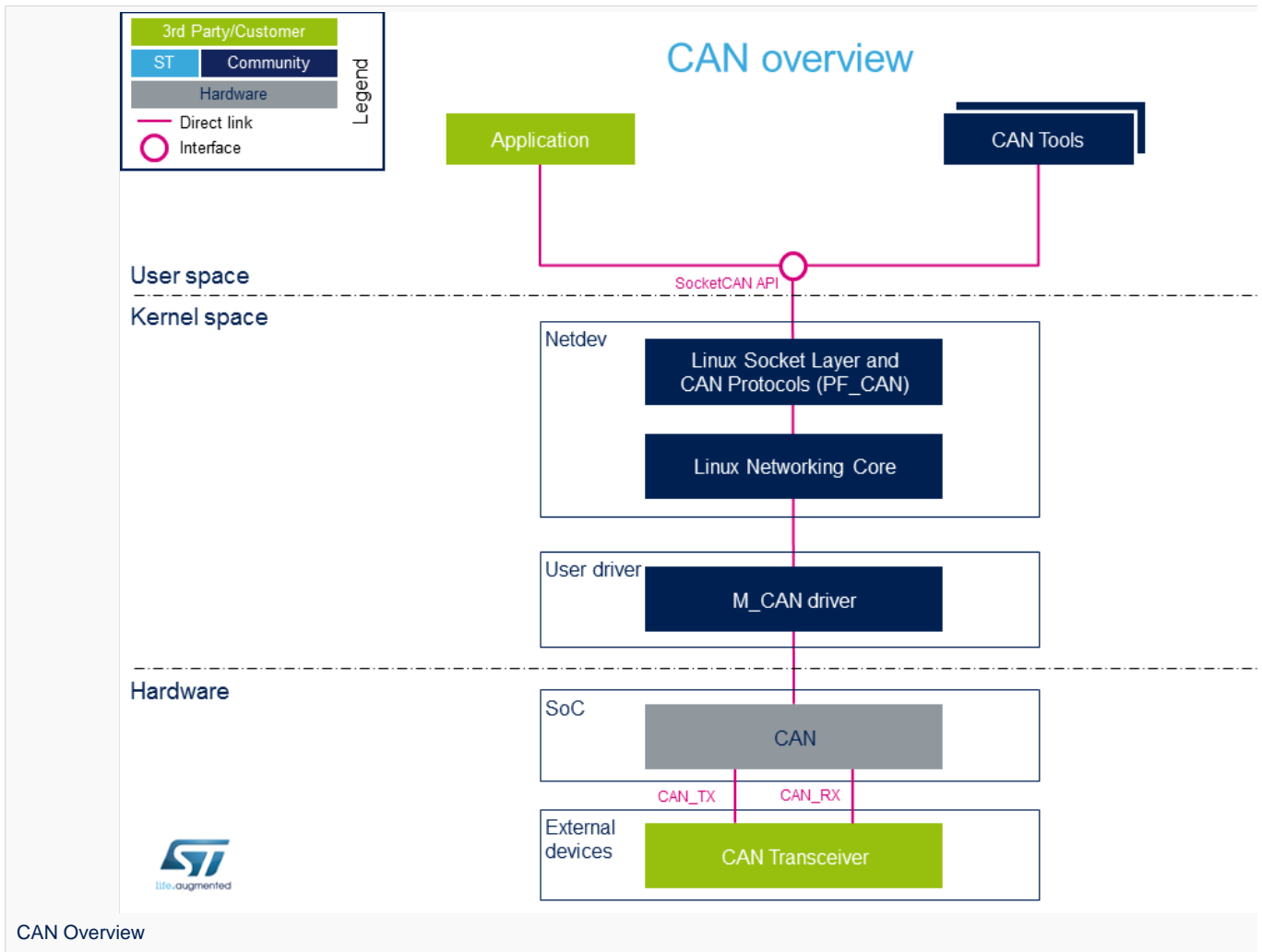
**SocketCAN**<sup>[3]</sup> is a uniform CAN Framework for the Linux kernel. It implements a new protocol family called **PF\_CAN**<sup>[4]</sup> and allows applications to receive and transmit CAN messages via Socket APIs with CAN specific socket options.

You can find many applications of CAN in the automotive industry. In vehicles, it allows electronic control units and devices to communicate with each other in applications without a host computer. For example, a high speed CAN bus is dedicated to security devices such as emergency brake system or airbags. Another low speed CAN bus is dedicated to comfort devices such as interior lighting or seat control.

This article describes the main components and APIs of the CAN Framework and gives examples of CAN usage.



## 2 System overview



### 2.1 Component description

*From user space to hardware*

- **Application** (User space)

Application to read/write data on the SocketCAN interface for communication with external devices connected on the CAN network (such as can-utils).

- **CAN tools** (User space)

Set of utilities for configuring and enabling SocketCAN interface (such as iproute2).

- **SocketCAN** (Kernel space)

Socket interface with specific CAN options which builds upon the Linux network layer.

- **Linux Socket Layer and CAN Protocols (PF\_CAN)** (Kernel space)



---

The protocol family, PF\_CAN<sup>[4]</sup>, provides an API for transport protocol modules to register and the structures to enable different CAN protocols on the bus.

- **Linux Networking Core** (Kernel space)

Kernel network layer that adapts the message to the transport protocol in use. The network subsystem of the Linux kernel is designed to be completely protocol-independent.

- **M\_CAN Driver** (Kernel space)

Driver implemented as a network interface for Bosch M\_CAN controller<sup>[5]</sup>.

- **CAN** (Hardware)

This is the CAN Core IP.

- **CAN Transceiver** (Hardware)

Interface between the CAN protocol controller and the physical wires of the CAN bus lines.

## 2.2 API description

The SocketCAN interface API description can be found in kernel documentation <sup>[3]</sup>.



## 3 Configuration

### 3.1 Kernel configuration

Activate the CAN driver in kernel configuration with Linux Menuconfig tool.

For compiling M\_CAN driver, select "Bosch M\_CAN devices":

```
[*] Networking support ---
>
  <*> CAN bus subsystem support --->
    CAN Device Drivers --->
      <*> Bosch M_CAN support
      <*> Bosch M_CAN support for io-mapped devices
```

M\_CAN driver is activated by default in ST deliveries.

### 3.2 Device tree configuration

CAN generic DT bindings:

- The M\_CAN device tree bindings<sup>[6]</sup> describe all the required and optional properties.

Detailed DT configuration for STM32 internal peripherals:

- [FDCAN device tree configuration](#)



---

## 4 How to use the framework

---

The CAN device must be configured via netlink interface. The following articles give user space examples of how to set up a SocketCAN interface (and configure settings like bit-timing parameters) and how to send/receive data on the CAN bus.

### 4.1 How to set up a SocketCAN interface

How to set up a SocketCAN interface

### 4.2 How to send/receive CAN data

How to send or receive CAN data



---

## 5 How to trace and debug the framework

---

### 5.1 How to trace

CAN Framework, specifically M\_CAN driver, print out info and error messages. You can display them with dmesg command:

```
Board $> dmesg | grep m_can
[   1.327824] m_can 4400e000.can: m_can device registered (irq=30, version=32)
[  25.560759] m_can 4400e000.can can0: bitrate error 0.3%
[  25.564630] m_can 4400e000.can can0: bitrate error 1.6%
```

### 5.2 How to monitor CAN bus

You can use the CAN FD adapter **PCAN-USB Pro FD**<sup>[7]</sup> to connect a computer to the CAN network via USB. The PCAN-View software provided with the tool is a monitoring program that allows to supervise the data flow on the CAN network and to detect frame errors.



---

## 6 Source code location

---

The source files are located inside the Linux kernel.

- **PF\_CAN:** af\_can.c<sup>[4]</sup>
- **M\_CAN driver:** m\_can.c<sup>[5]</sup>





---

## 7 To go further

---

CAN bit timing calculation plays an important role in ensuring performance of CAN network. To avoid transmission errors, the bit timing must be configured properly.

For more information about CAN bit timing:

- *Computation of CAN Bit Timing Parameters Simplified*<sup>[8]</sup>, from the CAN in Automation group (CiA)
- *The Configuration of the CAN Bit Timing*<sup>[9]</sup>, from Bosch documentation



## 8 References

- CAN FD - The basic idea, from the CAN in Automation group (CiA)
- CAN protocol implementations, from the CAN in Automation group (CiA)
- 3.03.1 Kernel SocketCAN documentation , Linux Foundation
- 4.04.14.2 net/can/af\_can.c , Protocol family CAN core module
- 5.05.1 drivers/net/can/m\_can/ , Driver for Bosch M\_CAN controller
- Documentation/devicetree/bindings/net/can/m\_can.txt M\_CAN device tree bindings
- PCAN-USB Pro FD description, by PEAK System
- Computation of CAN Bit Timing Parameters Simplified, from the CAN in Automation group (CiA)
- The Configuration of the CAN Bit Timing, from Bosch documentation

Stable: 01.12.2020 - 10:53 / Revision: 10.06.2020 - 14:35

A quality version of this page, approved on *1 December 2020*, was based off this revision.

### Contents

1 Article purpose .....	19
2 DT bindings documentation .....	20
3 DT configuration .....	21
3.1 DT configuration (STM32 level) .....	21
3.2 DT configuration (board level) .....	22
3.3 DT configuration examples .....	22
4 How to configure the DT using STM32CubeMX .....	23
5 References .....	24



---

## 1 Article purpose

---

This article explains how to configure the FDCAN when it is assigned to the Linux<sup>®</sup> OS. In that case, it is controlled by the CAN framework for Bosch M\_CAN controller.

The configuration is performed using the [device tree](#) mechanism that provides a hardware description of the FDCAN peripheral, used by the M\_CAN Linux driver and by the NET/CAN framework.

If the peripheral is assigned to another execution context, refer to [How to assign an internal peripheral to a runtime context](#) article for guidelines on peripheral assignment and configuration.



---

## 2 DT bindings documentation

---

M\_CAN device tree bindings<sup>[1]</sup> describe all the required and optional properties.



### 3 DT configuration

This hardware description is a combination of the **STM32 microprocessor** device tree files (*.dtsi* extension) and **board** device tree files (*.dts* extension). See the [Device tree](#) for an explanation of the device tree file split.

**STM32CubeMX** can be used to generate the board device tree. Refer to [How to configure the DT using STM32CubeMX](#) for more details.

#### 3.1 DT configuration (STM32 level)

All M\_CAN nodes are described in `stm32mp153.dtsi` <sup>[2]</sup> file with disabled status and required properties such as:

- Physical base address and size of the device register map
- Message RAM address and size (CAN SRAM)
- Host clock and CAN clock
- Message RAM configuration

This is a set of properties that may not vary for a given STM32 device.

```
m_can1: can@4400e000 {
    compatible = "bosch,m_can";
    reg = <0x4400e000 0x400>, <0x44011000 0x1400>;      /* FDCAN1 uses only the first
half of the dedicated CAN_SRAM */
    reg-names = "m_can", "message_ram";
    interrupts = <GIC_SPI 19 IRQ_TYPE_LEVEL_HIGH>,
                <GIC_SPI 21 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "int0", "int1";
    clocks = <&rcc CK_HSE>, <&rcc FDCAN_K>;
    clock-names = "hclk", "cclk";
    bosch,mram-cfg = <0x0 0 0 32 0 0 2 2>;
    status = "disabled";
};

m_can2: can@4400f000 {
    compatible = "bosch,m_can";
    reg = <0x4400f000 0x400>, <0x44011000 0x2800>;      /* The 10 Kbytes of the
CAN_SRAM are mapped */
    reg-names = "m_can", "message_ram";
    interrupts = <GIC_SPI 20 IRQ_TYPE_LEVEL_HIGH>,
                <GIC_SPI 22 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "int0", "int1";
    clocks = <&rcc CK_HSE>, <&rcc FDCAN_K>;
    clock-names = "hclk", "cclk";
    bosch,mram-cfg = <0x1400 0 0 32 0 0 2 2>;          /* Set mram-cfg offset to
write FDCAN2 data on the second half of the dedicated CAN_SRAM */
    status = "disabled";
};
```

The required and optional properties are fully described in the [bindings](#) files.

#### Warning

This device tree part is related to STM32 microprocessors. It must be kept as is, without being modified by the end-user.



## 3.2 DT configuration (board level)

Part of the device tree is used to describe the FDCAN hardware used on a given board. The DT node ("**m\_can**") must be filled in:

- Enable the CAN block by setting **status = "okay"**.
- Configure the pins in use via **pinctrl**, through **pinctrl-0** (default pins), **pinctrl-1** (sleep pins) and **pinctrl-names**.

## 3.3 DT configuration examples

The example below shows how to configure and enable FDCAN1 instance at board level:

```

&m_can1 {
    pinctrl-names = "default", "sleep";           /* configure pinctrl modes for
m_can1 */
    pinctrl-0 = <&m_can1_pins_a>;                 /* configure m_can1_pins_a as
default pinctrl configuration for m_can1 */
    pinctrl-1 = <&m_can1_sleep_pins_a>;          /* configure m_can1_sleep_pins_a as
sleep pinctrl configuration for m_can1 */
    status = "okay";                             /* enable m_can1 */
};

```



---

## 4 How to configure the DT using STM32CubeMX

---

The STM32CubeMX tool can be used to configure the STM32MPU device and get the corresponding platform configuration device tree files.

The STM32CubeMX may not support all the properties described in the above DT bindings documentation paragraph. If so, the tool inserts **user sections** in the generated device tree. These sections can then be edited to add some properties and they are preserved from one generation to another. Refer to STM32CubeMX user manual for further information.



## 5 References

Please refer to the following links for additional information:

- [Documentation/devicetree/bindings/net/can/m\\_can.txt](#) M\_CAN device tree bindings
- [arch/arm/boot/dts/stm32mp153.dtsi](#) , STM32MP153 device tree file

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 .....	25
2 Introduction .....	26
3 STM32CubeMX generated assignment .....	27
4 Manual assignment .....	29
4.1 TF-A .....	29
4.2 U-boot .....	29
4.3 Linux kernel .....	30
4.4 STM32Cube .....	30
4.5 OP-TEE .....	31





---

## 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 assignment 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

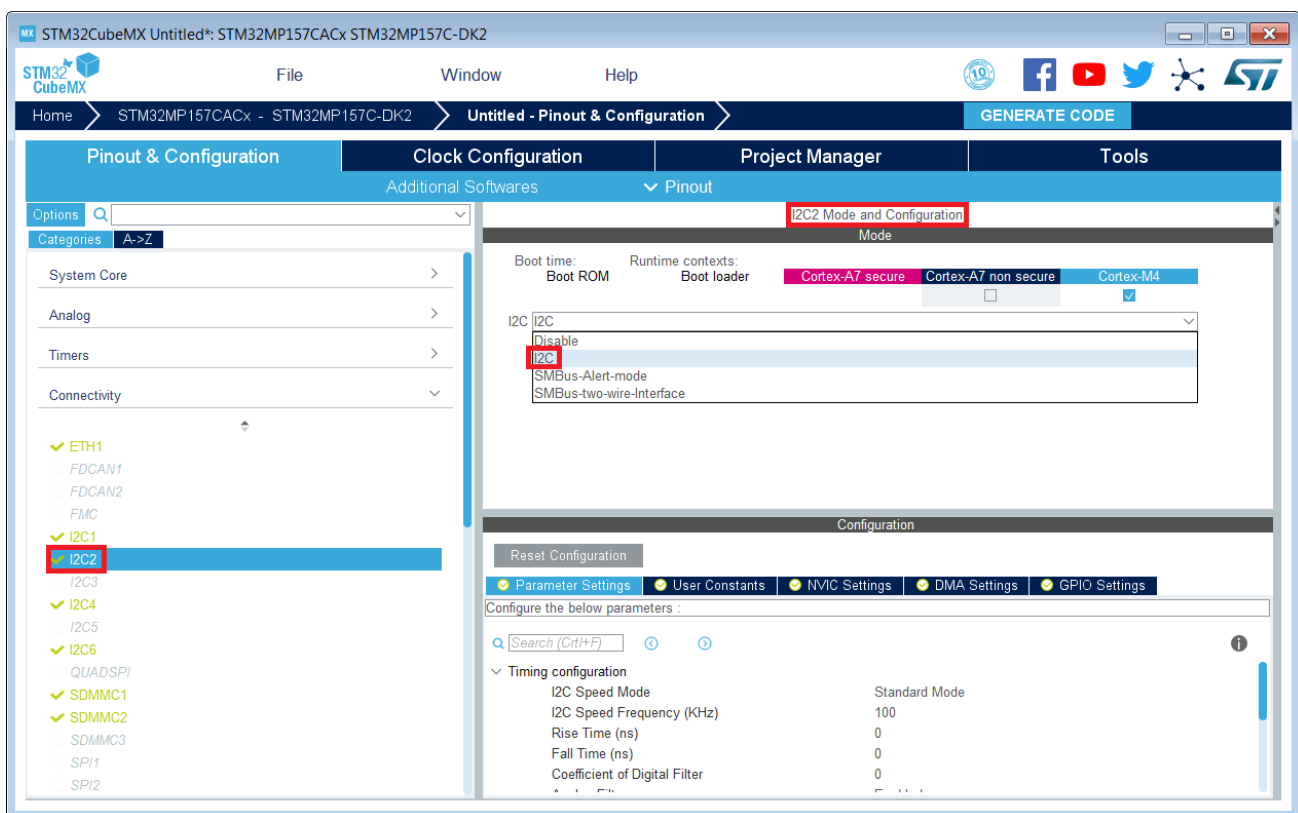
Beyond the peripherals assignment, explained in this article, it is also important to understand [How to configure system resources](#) (i.e clocks, regulator, gpio,...), shared between the Cortex-A7 and Cortex-M4 contexts



### 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.



### **i** 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";
};
```

### **i** 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 use in a secure context. This information is used to consolidate system configuration including secure hardening of configurable peripherals.

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

Das U-Boot -- the Universal Boot Loader (see [U-Boot\\_overview](#))

Stable: 31.03.2021 - 11:58 / Revision: 23.03.2021 - 14:07

A quality version of this page, approved on 31 March 2021, was based off this revision.

### Contents

1 Introduction .....	32
2 STM32Cube MP1 Package architecture .....	33
2.1 Level 0 (Drivers) .....	34
2.1.1 HAL drivers .....	34
2.1.1.1 HAL drivers overview .....	34
2.1.1.2 List of HAL drivers .....	34
2.1.2 LL drivers .....	35
2.1.2.1 Low Layer drivers overview .....	35
2.1.2.2 List of LL drivers .....	36
2.1.3 BSP drivers .....	36
2.1.3.1 BSP drivers overview .....	36
2.1.3.2 List of BSP drivers .....	36
2.2 Level 1 (Middlewares) .....	37
2.2.1 OpenAMP .....	37
2.2.2 FreeRTOS .....	37
2.3 Level 2 (Boards demonstrations) .....	38
2.4 Utilities .....	38
2.5 CMSIS .....	39
3 STM32Cube MP1 Package versus legacy STM32Cube MCU Package .....	42
4 References .....	43



---

## 1 Introduction

---

This article introduces **STM32Cube MP1 Package** architecture based on the Arm<sup>®</sup> Cortex<sup>®</sup>-M processor (e.g. Arm Cortex-M4)

- Please refer to [STM32Cube MP1 Package](#) article to get started.

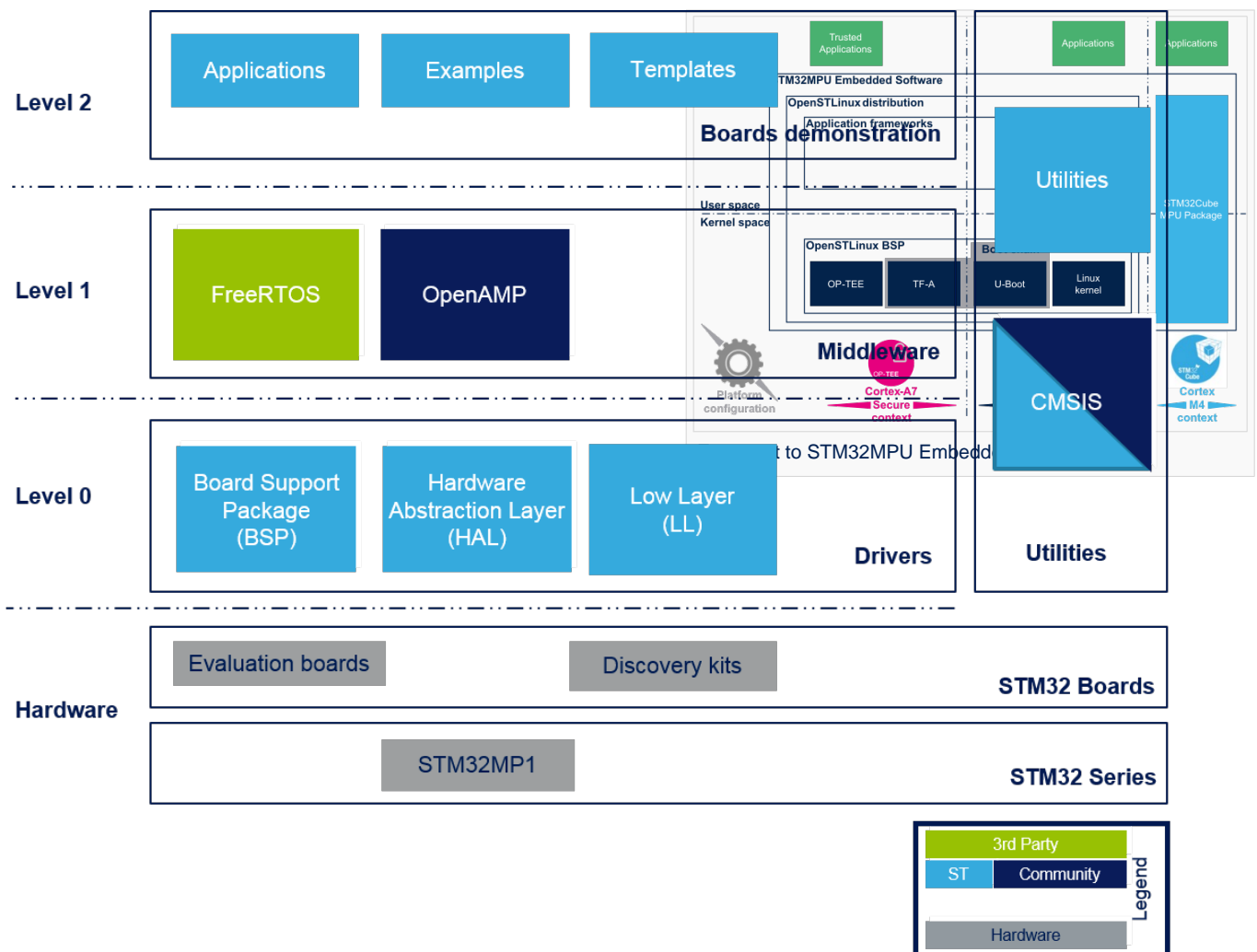




## 2 STM32Cube MP1 Package architecture

The **STM32Cube MP1 Package** gathers together, in a single package, all the generic embedded software components required to develop applications on top of Cortex-M microprocessors.

On top of the hardware, the **STM32Cube MP1 Package** solution is built around three levels of software components (Level 0 for Drivers, level 1 for Middlewares, Level 2 for Boards demonstrations), that interact easily with each other. It also includes 2 common components CMSIS and Utilities which interact with all three levels.



### Information

#### Notes:

- **HAL drivers** deal with the STM32 "internal" devices: they are related to the STM32MP15 internal peripherals
- **BSP drivers** deal with the boards configuration and high-level APIs: they are the equivalent of the Linux DT mechanism (Device tree or STM32MP15 device tree) and of the Linux frameworks (Linux application frameworks overview)



## 2.1 Level 0 (Drivers)

This level is divided into three software components:

- **Hardware Abstraction Layer (HAL)**
- **Low Layer (LL)**
- **Board Support Package (BSP)**

### 2.1.1 HAL drivers

The **HAL drivers** provide the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). They provide generic, multi instance and function-oriented APIs which simplify user application implementation by providing ready-to-use processes.

As example, for the communication peripherals (I2C, UART...), they include APIs allowing to initialize and configure the peripheral, to manage data transfer based on polling, interrupt or DMA process, and to handle communication errors that may raise during communication.

#### Information

##### Note:

- Please refer to [STM32MP15 reference manuals](#) to get detailed information about all supported IPs of STM32MP15xx family

#### 2.1.1.1 HAL drivers overview

The HAL APIs layer is composed of native and extended APIs set. It is directly built around a generic architecture and allows the build-upon layers, like the middleware layer, to implement its functions without in-depth knowledge about the used STM32 device. This improves the library code reusability and guarantees an easy portability on other devices and STM32 families

Contrary to the low layer drivers (see HAL Low Layer section), the HAL ones are functionality-oriented and not IP-oriented, Example: for the Timer peripheral, the APIs could be split into several categories following the functions offered by the IPs (Basic timer, capture, PWM ...etc.).

The HAL Drivers are a set of common APIs with a high compliancy level with most of the clients available on the market (stacks) called native APIs and embed also some extended functionalities for special services or a combination of several features offered by the STM32 peripherals

The HAL drivers APIs are split in two categories:

- Generic APIs which provide common and generic functions to all the STM32 Series
- Extension APIs which provide specific customized functions for a specific family or a specific part number

#### 2.1.1.2 List of HAL drivers

Please find hereafter the list of HAL drivers available for STM32MP1xx family :

#### Information

##### Note:

- Please refer to [List of HAL Drivers](#) to get full list of delivered HAL drivers

— stm32mp1xx_hal_adc.c	ADC HAL Driver
— stm32mp1xx_hal_adc_ex.c	ADC Extended HAL Driver
— stm32mp1xx_hal_cec.c	CEC HAL Driver
— stm32mp1xx_hal_cortex.c	CORTEX HAL Driver
— stm32mp1xx_hal_crc.c	CRC HAL Driver



stm32mp1xx_hal_crc_ex.c	CRC Extended HAL Driver
stm32mp1xx_hal_cryp.c	CRYP HAL Driver
stm32mp1xx_hal_cryp_ex.c	CRYP Extended HAL Driver
stm32mp1xx_hal_dac.c	DAC HAL Driver
stm32mp1xx_hal_dac_ex.c	DAC Extended HAL Driver
stm32mp1xx_hal_dcmi.c	DCMI HAL Driver
stm32mp1xx_hal_dfstm.c	DFSDM HAL Driver
stm32mp1xx_hal_dfstm_ex.c	DFSDM Extended HAL Driver
stm32mp1xx_hal_dma.c	DMA HAL Driver
stm32mp1xx_hal_dma_ex.c	DMA Extended HAL Driver
stm32mp1xx_hal_exti.c	EXTI HAL Driver
stm32mp1xx_hal_fdcan.c	FDCAN HAL Driver
stm32mp1xx_hal_gpio.c	GPIO HAL Driver
stm32mp1xx_hal_gpio_ex.c	GPIO Extended HAL Driver
stm32mp1xx_hal_hash.c	HASH HAL Driver
stm32mp1xx_hal_hash_ex.c	HASH Extended HAL Driver
stm32mp1xx_hal_hsem.c	HSEM HAL Driver
stm32mp1xx_hal_i2c.c	I2C HAL Driver
stm32mp1xx_hal_i2c_ex.c	I2C Extended HAL Driver
stm32mp1xx_hal_ipcc.c	IPCC HAL Driver
stm32mp1xx_hal_lptim.c	LPTIM HAL Driver
stm32mp1xx_hal_mdios.c	MDIOS HAL Driver
stm32mp1xx_hal_mdma.c	MDMA HAL Driver
stm32mp1xx_hal_pwr.c	PWR HAL Driver
stm32mp1xx_hal_pwr_ex.c	PWR Extended HAL Driver
stm32mp1xx_hal_qspi.c	QSPI HAL Driver
stm32mp1xx_hal_rcc.c	RCC HAL Driver
stm32mp1xx_hal_rcc_ex.c	RCC Extended HAL Driver
stm32mp1xx_hal_rng.c	RNG HAL Driver
stm32mp1xx_hal_rtc.c	RTC HAL Driver
stm32mp1xx_hal_rtc_ex.c	RTC Extended HAL Driver
stm32mp1xx_hal_sai.c	SAI HAL Driver
stm32mp1xx_hal_sai_ex.c	SAI Extended HAL Driver
stm32mp1xx_hal_sd.c	SD HAL Driver
stm32mp1xx_hal_smartcard.c	SMARTCARD HAL Driver
stm32mp1xx_hal_smartcard_ex.c	SMARTCARD Extended HAL Driver
stm32mp1xx_hal_smbus.c	SMBUS HAL Driver
stm32mp1xx_hal_spdifrx.c	SPDIFRX HAL Driver
stm32mp1xx_hal_spi.c	SPI HAL Driver
stm32mp1xx_hal_spi_ex.c	SPI Extended HAL Driver
stm32mp1xx_hal_sram.c	FMC HAL Driver (for PSRAM)
stm32mp1xx_hal_tim.c	TIM HAL Driver
stm32mp1xx_hal_tim_ex.c	TIM Extended HAL Driver
stm32mp1xx_hal_uart.c	UART HAL Driver
stm32mp1xx_hal_uart_ex.c	UART Extended HAL Driver
stm32mp1xx_hal_usart.c	USART HAL Driver
stm32mp1xx_hal_usart_ex.c	USART Extended HAL Driver
stm32mp1xx_hal_wwdg.c	WWDG HAL Driver

## 2.1.2 LL drivers

The **Low Layer (LL) drivers** offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. The LL APIs are available only for a set of peripherals

### 2.1.2.1 Low Layer drivers overview

The Low Layer (LL) drivers are part of the STM32Cube firmware HAL that provides basic set of optimized and one-shot services. The Low layer drivers, contrary to the HAL ones are not Fully Portable across the STM32 families; the availability of some functions depends on the physical availability of the relative features on the product.

The Low Layer (LL) drivers are designed to offer the following features:

- New set of inline functions for direct and atomic register access
- One-shot operations that can be used by the HAL drivers or from application level.



- Fully independent from HAL and can be used in standalone usage (without HAL drivers)
- Full feature coverage of all the supported peripherals

### 2.1.2.2 List of LL drivers

Please find hereafter the list of LL drivers available for STM32MP1xx family :

#### Information

##### Note:

- Please refer to [List of LL Drivers](#) to get full list of delivered LL drivers

stm32mp1xx_ll_adc.h	ADC LL Driver
stm32mp1xx_ll_bus.h	BUS LL Driver
stm32mp1xx_ll_cortex.h	CORTEX LL Driver
stm32mp1xx_ll_dma.h	DMA LL Driver
stm32mp1xx_ll_dmamux.h	DMAMUX LL Driver
stm32mp1xx_ll_exti.h	EXTI LL Driver
stm32mp1xx_ll_gpio.h	GPIO LL Driver
stm32mp1xx_ll_hsem.h	HSEM LL Driver
stm32mp1xx_ll_i2c.h	I2C LL Driver
stm32mp1xx_ll_ipcc.h	IPCC LL Driver
stm32mp1xx_ll_lptim.h	LPTIM LL Driver
stm32mp1xx_ll_pwr.h	PWR LL Driver
stm32mp1xx_ll_rcc.h	RCC LL Driver
stm32mp1xx_ll_rtc.h	RTC LL Driver
stm32mp1xx_ll_spi.h	SPI LL Driver
stm32mp1xx_ll_sram.h	FMC LL Driver (for PSRAM)
stm32mp1xx_ll_system.h	SYSTEM LL Driver (SYSCFG & DBGMCU)
stm32mp1xx_ll_tim.h	TIM LL Driver
stm32mp1xx_ll_usart.h	USART LL Driver
stm32mp1xx_ll_utils.h	UTILITIES LL Driver
stm32mp1xx_ll_wwdg.h	WWDG LL Driver

### 2.1.3 BSP drivers

The **BSP drivers** are firmware components based on the HAL drivers and provide a set of APIs relative to the hardware components in the evaluation boards coming with the **STM32Cube Package**. All examples and applications given with the **STM32Cube** are based on these BSP drivers.

#### 2.1.3.1 BSP drivers overview

The BSP architecture proposes a new model that prevents some Standard library weaknesses and provides more features:

- Portable external resources code (components): the external components could be used by all STM32 families.
- Multiple use of hardware resources without duplicated initialization: example: I2C Physical Layer could be used for several EVAL Drivers
- Intuitive functionalities based on high level use case
- Portable BSP drivers for different external devices

#### 2.1.3.2 List of BSP drivers

The **BSP drivers** offer a set of APIs relative to the hardware components available in the hardware boards (LEDs, Buttons and COM port).



```

STM32MP15xx_DISCO
├── bumpversion.cfg
├── Release_Notes.html
├── STM32MP15xx_DISCO_BSP_User_Manual.chm
├── stm32mp15xx_disco_bus.c
├── stm32mp15xx_disco_bus.h
├── stm32mp15xx_disco.c
├── stm32mp15xx_disco_conf_template.h
├── stm32mp15xx_disco_errno.h
├── stm32mp15xx_disco.h
├── stm32mp15xx_disco_stpmic1.c
├── stm32mp15xx_disco_stpmic1.h
├── STM32MP15xx_EVAL
│   ├── bumpversion.cfg
│   ├── Release_Notes.html
│   ├── STM32MP15xx_EVAL_BSP_User_Manual.chm
│   ├── stm32mp15xx_eval_bus.c
│   ├── stm32mp15xx_eval_bus.h
│   ├── stm32mp15xx_eval.c
│   ├── stm32mp15xx_eval_conf_template.h
│   ├── stm32mp15xx_eval_errno.h
│   ├── stm32mp15xx_eval.h
│   ├── stm32mp15xx_eval_stpmic1.c
│   └── stm32mp15xx_eval_stpmic1.h

```

## 2.2 Level 1 (Middlewares)

Middleware components are a set of libraries providing a set of services. **STM32Cube MP1 Package** offers 2 main components : **OpenAMP** and **FreeRTOS**

Each middleware component is mainly composed of:

- Library core: this is the core of a component; it manages the main library state machine and the data flow between the several modules.
- Interface layer: the interface layer is generally used to link the component core with the lower layers like the HAL and the BSP drivers

### 2.2.1 OpenAMP

OpenAMP is a library implementing the Remote Processor Service framework (RPMsg) which is a messaging mechanism to communicate with a remote processor.

- Load and control Cortex<sup>®</sup>-M firmware
- Inter processor communication

#### Information

Note:

- Please refer to [Coprocessor\\_management\\_overview](#) to get more information related to coprocessor management

### 2.2.2 FreeRTOS

FreeRTOS is a Free Real Time Operating System (RTOS). The **FreeRTOS** offers preemptive real-time performance with optimized context switch and interrupt times, enabling fast, highly predictable response times.



It includes the following main features :

- Small memory footprint
- High portability
- Multithread management
- Pre-emptive scheduling
- Fast interrupt response
- Extensive inter-process communication
- Synchronization facilities
- Tickless operation during low-power mode
- Open source standard
- CMSIS compatibility layer

### 2.3 Level 2 (Boards demonstrations)

The **Boards demonstrations** level is composed of a single layer which provides all Examples and Applications. It includes also all **STM32CubeIDE** projects for each supported board as well as Templates source files.

There are 4 kinds of projects demonstrating different usages of software APIs from level 0 (Drivers) and level 1 (Middleware):

- **Examples projects** showing how to use HAL APIs and Low Layer drivers if any (Level 0) with very basic usage of BSP layer (buttons and LEDs in general)
- **Applications projects** showing how to use the middleware components (Level 1) and how to integrate them with the hardware and BSP/HAL layers (Level 0). These applications could be hybrid and use several other middleware components.
- **Demonstrations projects** showing how to integrate and run a maximum number of peripherals and Middleware stacks to showcase the product features and performance
- **Templates projects** is a really basic user application including IDE projects files, which could be used to start a custom project

#### Information

##### Notes:

- Please refer to [STM32Cube MP1 Package Overview](#) to get information on locating Examples, Applications and Demonstrations in **STM32Cube MP1 Package**
- Please refer to [List of projects](#) to get information on the list of available Examples, Applications and Demonstrations in **STM32Cube MP1 Package**

### 2.4 Utilities

The **Utilities** is a set of common utilities and services offered by **STM32Cube MP1 Package** and is composed of different components :

```
└─ Utilities
  └─ ResourceManager    Services for coprocessing in multi-core devices. Refer to Reso
  urce_manager_for_coprocessing
```



## 2.5 CMSIS

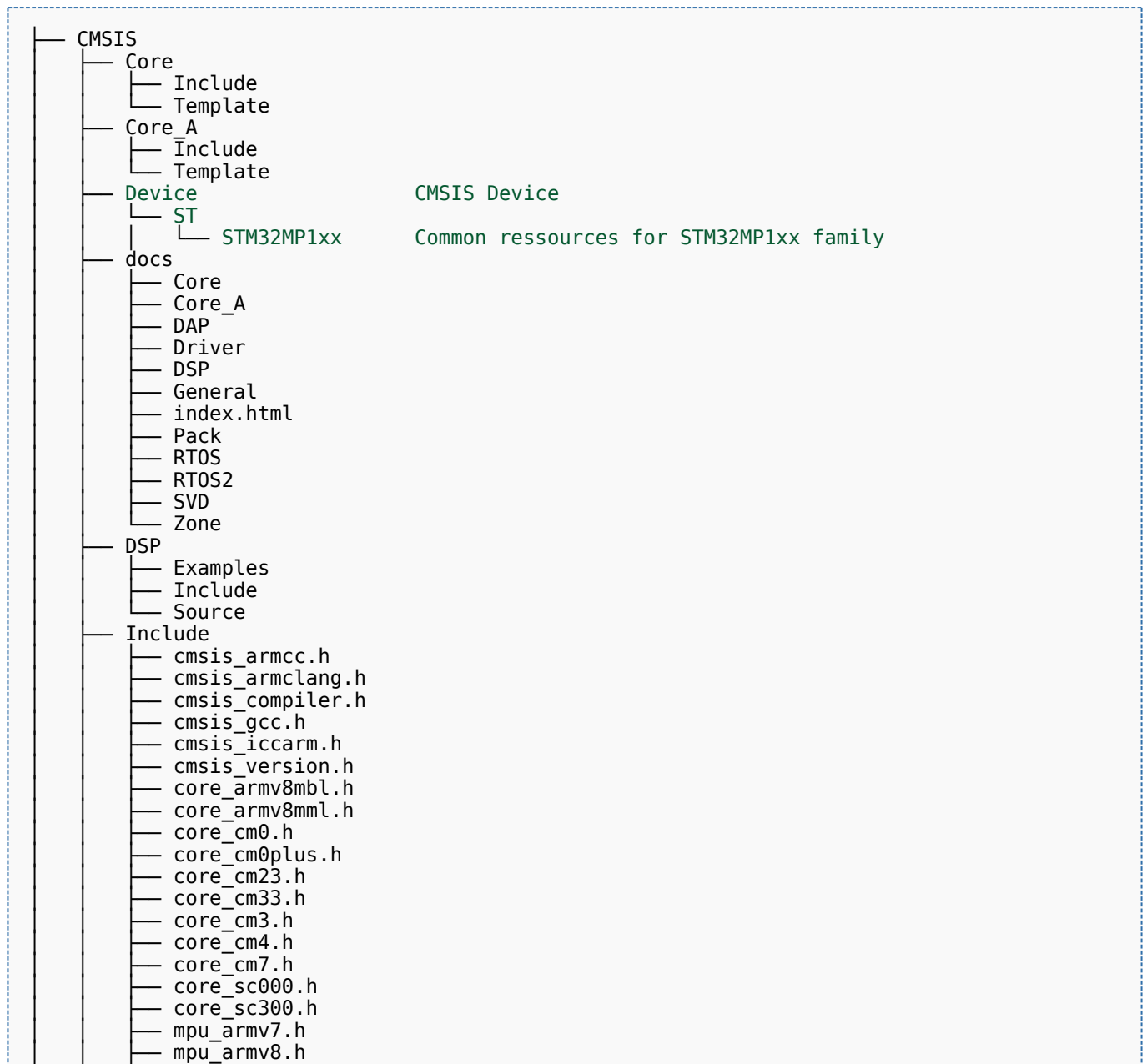
The **Cortex Microcontroller Software Interface Standard** (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series.

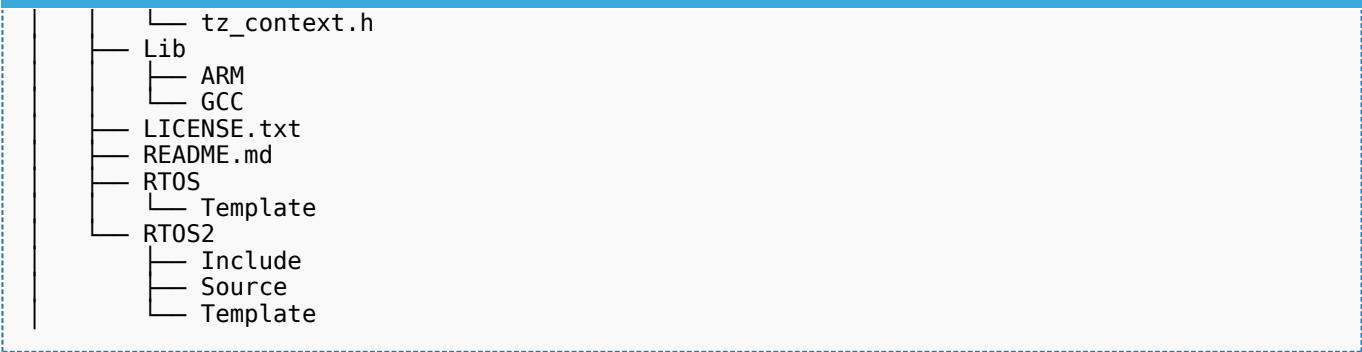
- Please refer to article [CMSIS](#) to get more information on CMSIS component

The CMSIS component also provides specific common resources for device support. It enables consistent and simple software interfaces to the processor and the peripherals, simplifying software re-use, reducing the learning curve for microcontroller developers, and reducing the time to market for new devices

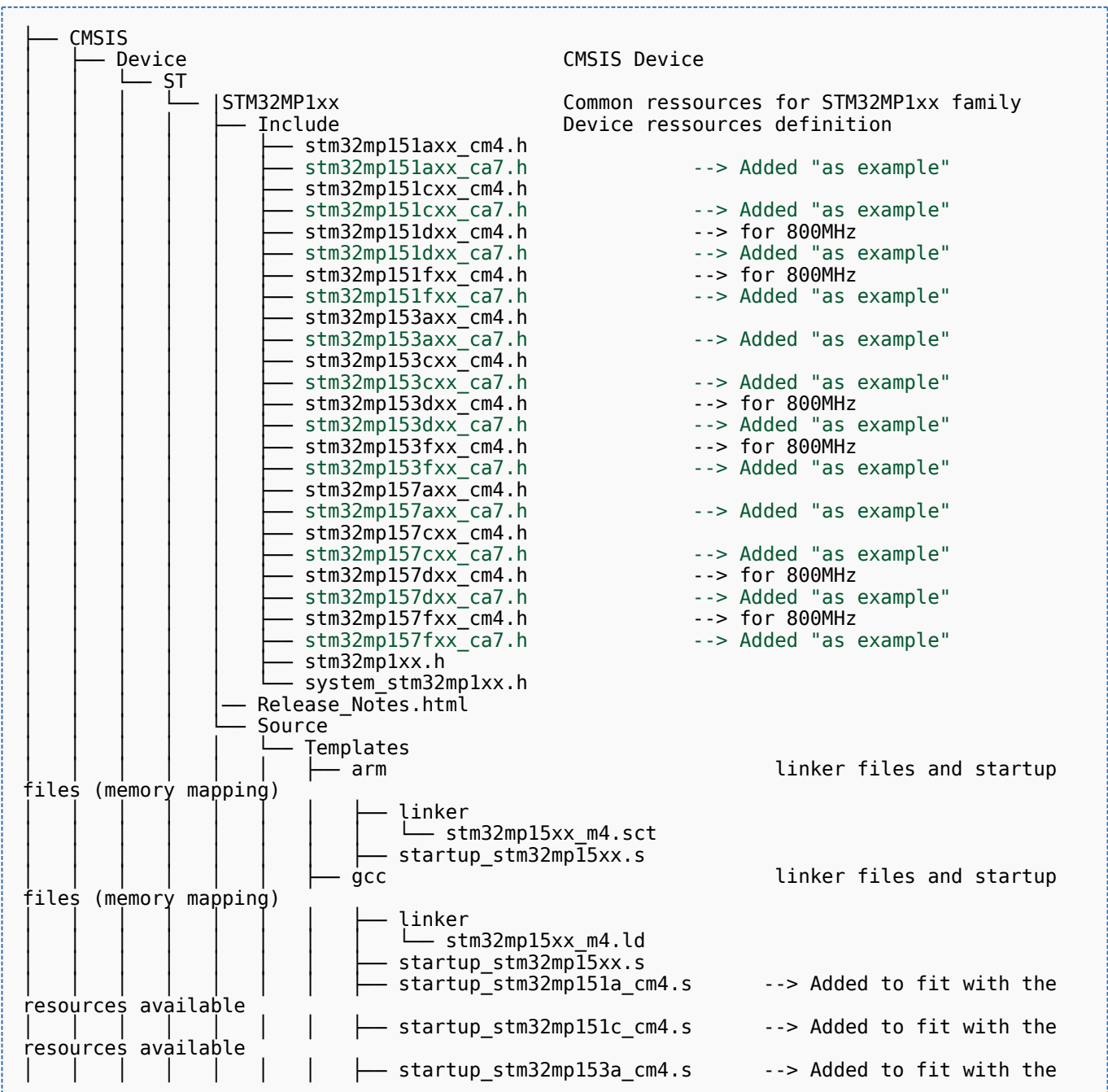
This vendor part is called **CMSIS Device** and it provides interrupt list, peripherals registers description and associated defines for all registers bit fields.

- **CMSIS** structure overview:





• CMSIS Device structure :









### 3 STM32Cube MP1 Package versus legacy STM32Cube MCU Package

STM32 MPU devices introduce light differences with STM32 MCU. So please find hereafter a short description of the main differences between STM32Cube MP1 Package and STM32Cube MCU Package:

- The middleware and BSP components offer is smaller in STM32Cube MP1 Package as we can take advantage of a rich OS like Linux<sup>®</sup> running on Cortex-A core for networking, USB, visual and audio services

#### Information

##### Notes:

- All Middlewares provided by STM32Cube MCU Package should be compatible with MPU environment even if not provided in STM32Cube MP1 Package (it means they are not tested)
- All BSP components provided by STM32Cube MCU Package are not compatible with MPU environment as they are managed by Linux OS on main processor Cortex A
- There is no Flash HAL driver as there is no volatile embedded FLASH dedicated to Cortex-M in MPU devices
- Specific pieces of software have been added to handle multi-core operations:
  - OpenAMP middleware for Intercommunication processeur between cortex A and cortex M (RPMsg protocol implementation)
  - Resource Manager library for system resource management
  - Virtual UART driver (specific usage when Linux is used on Cortex-A)
  - Linux script to load STM32Cube MPU firmware running on Cortex-M core (specific usage when Linux is used on Cortex-A)

#### Information

##### Note:

- Please refer to [Getting\\_started\\_with\\_STM32\\_MPU\\_devices](#) article to get an overview of STM32 MPU devices



---

## 4 References

---

Universal Asynchronous Receiver/Transmitter

Universal Synchronous/Asynchronous Receiver/Transmitter

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 tuning 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: 17.11.2021 - 16:41 / Revision: 17.11.2021 - 10:47

A quality version of this page, approved on 17 November 2021, 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
<b>Application notes</b>			
AN4803	High-speed SI simulations using IBIS and board-level simulations using HyperLynx® SI on STM32 MCUs and MPUs	<a href="#">AN4803.pdf</a>	v2.0
AN5027	Interfacing PDM digital microphones using STM32 MCUs and MPUs	<a href="#">AN5027.pdf</a>	v2.0
AN5031	Getting started with STM32MP15 Series hardware development	<a href="#">AN5031.pdf</a>	v3.0
AN5036	Thermal management guidelines for STM32 applications	<a href="#">AN5036.pdf</a>	v3.0
AN5109	STM32MP1 Series using low-power modes	<a href="#">AN5109.pdf</a>	v4.0
AN5122	STM32MP1 Series DDR memory routing guidelines	<a href="#">AN5122.pdf</a>	v3.0
AN5168	STM32MP1 series DDR configuration	<a href="#">AN5168.pdf</a>	 v2.0
	USB Type-C™ Power Delivery using STM32xx Series MCUs and	<a href="#">AN522</a>	










Reference	Name	Link	Version
<b>Application notes</b>			
AN5225	STM32xxx Series MPUs	5.pdf	 v5.0
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	v2.0
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 guidelines	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
<b>Datasheets<sup>[1]</sup></b>			
DS12505	STM32MP157C/F datasheet (secure)	DS12505.pdf	 v6.0
DS12504	STM32MP157A/D datasheet (basic)	DS12504.pdf	 v6.0
DS12503	STM32MP153C/F datasheet (secure)	DS12503.pdf	 v6.0
DS12502	STM32MP153A/D datasheet (basic)	DS12502.pdf	 v6.0
DS12501	STM32MP151C/F datasheet (secure)	DS12501.pdf	 v6.0
DS12500	STM32MP151A/D datasheet (basic)	DS12500.pdf	 v6.0
DS12792	STPMIC1 datasheet	DS12792.pdf	 v8.0



Reference	Name	Link	Version
<b>Application notes</b>			
<b>Errata sheets</b>			
ES0438	STM32MP15xx device errata	<a href="#">ES0438.pdf</a>	v6.0
<b>Reference manuals<sup>[1]</sup></b>			
RM0436	STM32MP157 reference manual (STM32MP157xxx advanced Arm <sup>®</sup> -based 32-bit MPUs)	<a href="#">RM0436.pdf</a>	v5.0
RM0442	STM32MP153 reference manual (STM32MP153xxx advanced Arm <sup>®</sup> -based 32-bit MPUs)	<a href="#">RM0442.pdf</a>	v5.0
RM0441	STM32MP151 reference manual (STM32MP151xxx advanced Arm <sup>®</sup> -based 32-bit MPUs)	<a href="#">RM0441.pdf</a>	v5.0
<b>Boards schematics</b>			
MB1262 schematics	STM32MP157C-EV1 motherboard schematics MB1262-C01 board schematic (Evaluation board)	<a href="#">MB1262-C01.pdf</a>	v1.0
MB1263 schematics	STM32MP157F-EV1 daughterboard schematics MB1263-C04 board schematic (Evaluation board)	<a href="#">MB1263-C04.pdf</a>	v4.0
MB1230 schematics	DSI 720p LCD display daughterboard schematics MB1230-C board schematic (Evaluation board)	<a href="#">MB1230-C.pdf</a>	v1.1
MB1379 schematics	Camera daughterboard schematics MB1379-A01 board schematic (Evaluation board)	<a href="#">MB1379-A01.pdf</a>	v1.0
MB1272 schematics	STM32MP157x-DKx motherboard schematics MB1272-DK2-C01 board schematic (Discovery kit)	<a href="#">MB1272-C01.pdf</a>	v1.0
MB1407 schematics	STM32MP157x-DKx daughterboard schematics MB1407-LCD-C01 board schematic (Discovery kit)	<a href="#">MB1407-C01.pdf</a>	v1.0
<b>Boards user manuals</b>			
UM2535	STM32MP157x-EV1 evaluation board user manual	<a href="#">UM2535.pdf</a>	v2.0
UM2534	STM32MP157x-DKx discovery board user manual	<a href="#">UM2534.pdf</a>	v1.0
<b>Tools user manuals</b>			
UM2563	STM32CubeIDE installation guide	<a href="#">UM2563.pdf</a>	 v3.0





Reference	Name	Link	Version
<b>Application notes</b>			
UM2579	Migration guide from System Workbench to STM32CubeIDE	UM2579.pdf	v1.0
UM2553	STM32CubeIDE quick start guide	UM2553.pdf	 v3.0
AN5360	Getting started with projects based on the STM32MP1 Series in STM32CubeIDE	AN5360.pdf	v1.0
UM2609	STM32CubeIDE user guide	UM2609.pdf	 v5.0
UM1718	STM32CubeMX user manual	UM1718.pdf	 v36.0
UM2237	STM32CubeProgrammer tool user manual	UM2237.pdf	 v17.0
UM2238	STM32 Trusted Package Creator tool user manual	UM2238.pdf	 v9.0
UM2542	STM32 Series Key Generator tool user manual	UM2542.pdf	 v2.0
UM2543	STM32 Series Signing tool user manual	UM2543.pdf	 v2.0

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



## Archives

STM32MP15 release	ST documentation
STM32MP15-Ecosystem-v3.0.0	<a href="#">STM32MP15 resources - v3.0.0</a> page for the previous v3 ecosystem release
STM32MP15-Ecosystem-v2.1.0	<a href="#">STM32MP15 resources - v2.1.0</a> page for the v2 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v2.0.0	<a href="#">STM32MP15 resources - v2.0.0</a> page for the v2 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.2.0	<a href="#">STM32MP15 resources - v1.2.0</a> page for the v1 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.1.0	<a href="#">STM32MP15 resources - v1.1.0</a> page for the v1 ecosystem releases (in archived wiki)
STM32MP15-Ecosystem-v1.0.0	<a href="#">STM32MP15 resources - v1.0.0</a> page for the v1 ecosystem releases (in archived wiki)

USB port or connector

Universal Synchronous/Asynchronous Receiver/Transmitter

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.

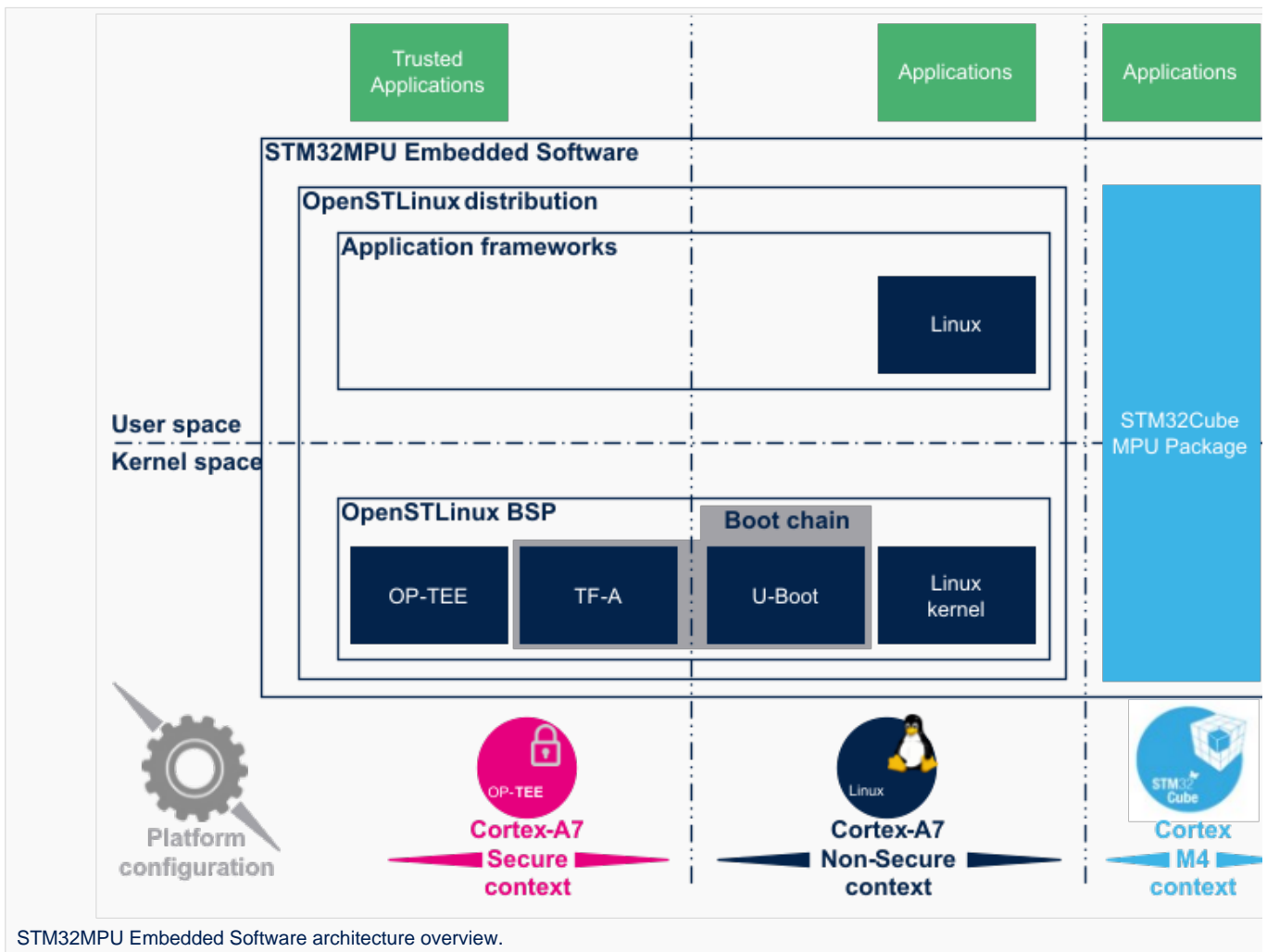


## 1 STM32MPU Embedded Software overview

The diagram below shows STM32MPU Embedded Software distribution main components:

- The **OpenSTLinux distribution**, running on the Arm<sup>®</sup> Cortex<sup>®</sup>-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<sup>®</sup> Cortex<sup>®</sup>-A in secure mode.
    - The **Linux<sup>®</sup> kernel** running on the Arm<sup>®</sup> Cortex<sup>®</sup>-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<sup>®</sup> Cortex<sup>®</sup>-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		Legend
ST	Community	



---

## 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<sup>[1]</sup>.

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<sup>[2]</sup>, until it is merged in the targeted repository.



---

### 3 References

---

- [https://en.wikipedia.org/wiki/Open-source\\_software](https://en.wikipedia.org/wiki/Open-source_software)
- STM32MP1 Distribution Package