



CRC internal peripheral



Contents

1. CRC internal peripheral	3
2. Crypto API overview	9
3. How to assign an internal peripheral to a runtime context	18
4. STM32CubeMP1 architecture	25
5. STM32CubeMX	39
6. STM32MP15 resources	42
7. STM32MPU Embedded Software architecture overview	46



A quality version of this page, approved on *12 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



1 Article purpose

The purpose of this article is to

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



2 Peripheral overview

The **CRC** peripheral is used to verify data transmission or storage integrity.

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

CRC1 and CRC2 are **non secure** peripherals.



3 Peripheral usage and associated software

3.1 Boot time

CRC instances are not used at boot time.

3.2 Runtime

3.2.1 Overview

CRC instances can be allocated to:

- the Arm®Cortex®-A7 non-secure for using in Linux® with Linux Crypto framework
- or
- the Arm®Cortex®-M4 for using in STM32Cube with STM32Cube CRC driver

Chapter Peripheral assignment describes which peripheral instance can be assigned to which context.

3.2.2 Software frameworks

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

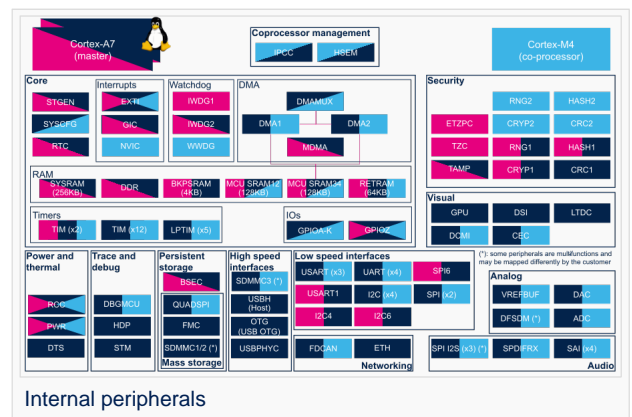
3.2.3 Peripheral configuration

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.

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)		
Security	CRC	CRC1			
		CRC2			



4 How to go further

Not applicable.



5 References

Cyclic redundancy check calculation unit

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



Cortex[®]

Linux[®] is a registered trademark of Linus Torvalds.

Open Portable Trusted Execution Environment

Stable: 19.10.2020 - 09:54 / Revision: 19.10.2020 - 09:52

A quality version of this page, approved on 19 October 2020, was based off this revision.

The Crypto API is a cryptography framework in the Linux[®] kernel. It is dedicated to the parts of the kernel that deal with cryptography, such as IPsec and dm-crypt.

Contents

1 Framework purpose	10
2 System overview	11
2.1 Description of the components	11
2.2 API description	12
3 Configuration	13
3.1 Kernal configuration	13
3.2 Devicetree configuration	13
4 How to use the Crypto API framework	14
5 Use cases	15
6 How to trace and debug the framework	16
6.1 How to monitor	16
6.2 How to trace	16
6.3 How to debug	16
7 Generic source code location	17
8 References	18



1 Framework purpose

The purpose of this article is to introduce the Crypto API framework:

- general information
- main component/stakeholders
- how to use the Crypto API
- use cases

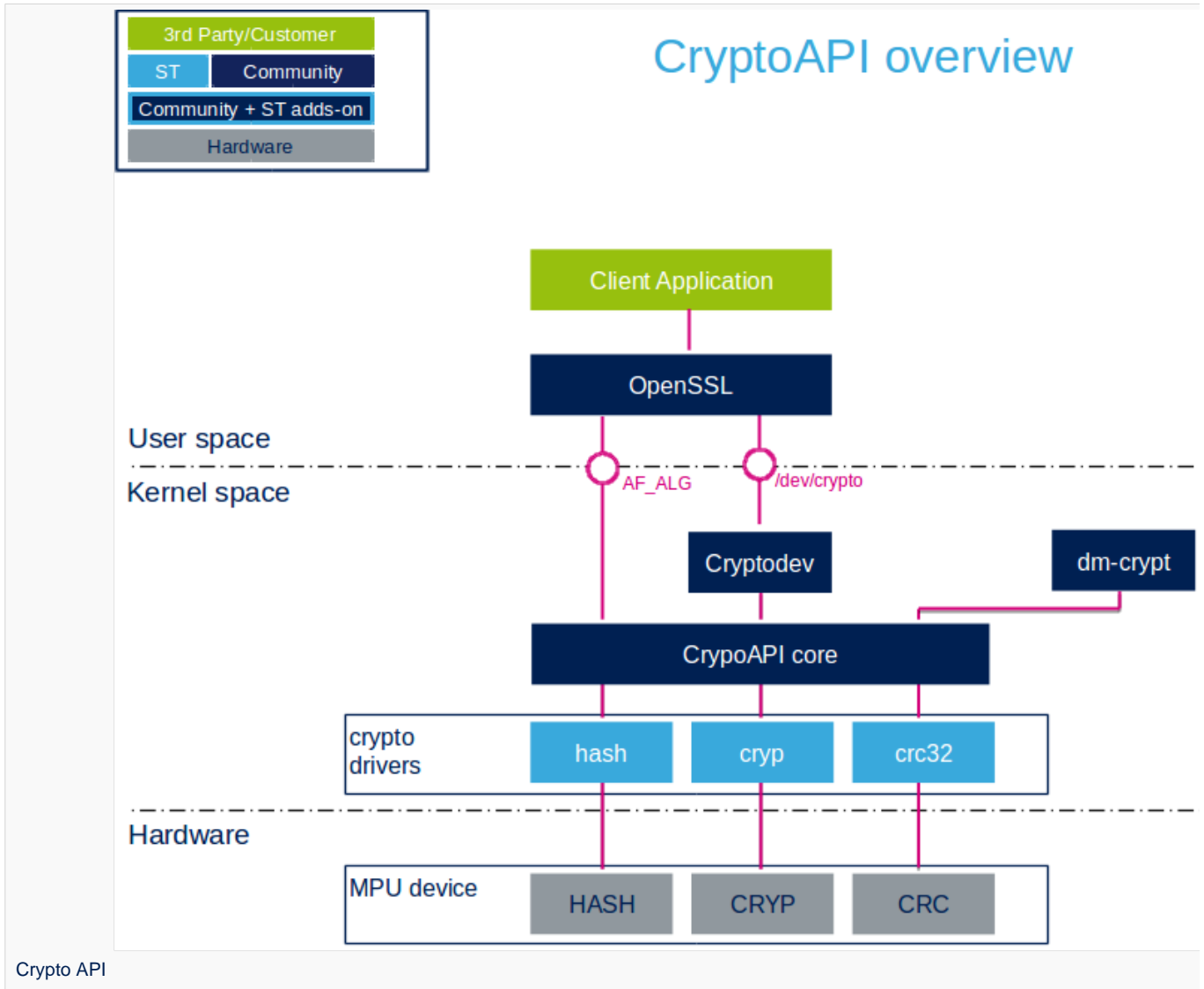
The Crypto API framework mainly includes all popular **hash** and **block ciphers** (encryption) functions.

A **hash** is a string or number generated from a text string. The length of the resulting string or number is fixed and widely varies with small variations of the input. The best hashing algorithms are designed so that it is impossible to turn a hash back into its original string. Hashing is particularly useful to compare a value with a stored value. However it cannot store its plain representation for security reasons. This makes hashing an ideal solution to store passwords.

Encryption turns data into a series of unreadable characters which length is not fixed. The encrypted strings can be reversed back into their original decrypted form if the right key is provided. Encrypting a confidential file is a good way to prevent anyone from accessing its content.

Drivers for CRYPT (block cipher), HASH (hash) and CRC (cyclic redundancy check) are integrated within the Crypto API kernel service.

2 System overview



2.1 Description of the components

i Information

OpenSSL and dm-crypt are not part of the Crypto API framework but they are typical users of the Crypto API services.

From User space to hardware

- **OpenSSL** (User space)



OpenSSL^[1] is a software library supporting the TLS and SSL protocols as well as cryptographic functions. Openssl is available in OpenSTLinux distribution.

- **dm-crypt** (Kernel space)

dm-crypt^[2] is a kernel disk encryption subsystem. It is natively available in the standard Linux kernel.

- **Cryptodev** (Kernel space)

Cryptodev^[3] is a device driver which provides a general interface for userland applications. Although it is not part of the standard Linux kernel, it is available in OpenSTLinux distribution.

- **CryptoAPI core** (Kernel space)

This layer represents the standard Linux kernel cryptographic framework.

- **hash, crypt and crc32** (Kernel space)

These are the cryptographic Linux drivers handling the internal peripherals.

- **HASH, CRYPT and CRC** (Hardware)

These HW blocks handle hash, ciphering, and CRC checksum.

2.2 API description

The Crypto API is documented in the Linux Kernel Crypto API section of the Linux Kernel documentation^[4]. It offers both a kernel and a userland interface:

- kernel internal interface, used in particular by dm-crypt.
- userland algorithm interface (socket) named AF_ALG^[5]. OpenSSL can use this interface.

In addition to the socket user interface, a more friendly interface, the cryptodev, can be used. It offers the /dev/crypto ioctl API. It is roughly described by the cryptodev.h^[6] header file. OpenSSL can be configured to use this interface as an alternative to the historical AF_ALG interface.



3 Configuration

3.1 Kernel configuration

The Crypto API is activated by default in ST deliveries. Nevertheless, if a specific configuration is required, you can use Linux Menuconfig tool: [Menuconfig](#) or [how to configure kernel](#) and select:

```
[*] Cryptographic API --->
  [*]   Hardware crypto devices --->
    [*]   Support for STM32 crc accelerators
    [*]   Support for STM32 hash accelerators
    [*]   Support for STM32 crypto accelerators
```

3.2 Devicetree configuration

By default the drivers are not enabled, so this needs to be added if you want to use HW accelerators.

- crc: CRC_device_tree_configuration.
- hash: HASH_device_tree_configuration.
- crypto: CRYPT_device_tree_configuration.



4 How to use the Crypto API framework

The Crypto API framework can be used by other kernel modules.

The Crypto API documentation provides kernel code examples^[7]:

- Symmetric-key cipher operation.
- Operational state memory with SHASH.



5 Use cases

- Disk encryption

This is a typical example of Crypto API framework usage. Refer to LUKS^[8] for a standard disk encryption process.



6 How to trace and debug the framework

6.1 How to monitor

The list of available ciphers is given in /proc/crypto:

```
Board $> cat /proc/crypto
```

Output part showing that an STM32 driver provides with the CRC32 cipher:

```
...
name      : crc32
driver    : stm32-crc32
module    : kernel
priority  : 200
refcnt    : 1
selftest  : passed
internal  : no
type      : shash
blocksize : 1
digestsize : 4
...
```

6.2 How to trace

There are no specific traces for this framework.

6.3 How to debug

There are no specific debug means for this framework.



7 Generic source code location

- CryptoAPI core
- CryptoAPI interface
- stm32 crypto drivers



8 References

- OpenSSL a software library supporting the TLS and SSL protocols as well as cryptographic functions.
- dm-crypt a kernel disk encryption subsystem
- Cryptodev a device driver which provides a general interface for userland applications
- Linux Kernel Crypto API the official crypto API kernel documentation
- Crypto API Userland interface specification of the userland API
- cryptodev.h cryptodev header file specifying the userland API
- Crypto API kernel code examples some kernel code examples using the Crypto API framework
- LUKS (Linux Unified Key Setup) a disk encryption specification

Application programming interface

Linux[®] is a registered trademark of Linus Torvalds.

Cryptographic processor

Cyclic redundancy check calculation unit

GPIO alternate function

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	19
2 Introduction	20
3 STM32CubeMX generated assignment	21
4 Manual assignment	23
4.1 TF-A	23
4.2 U-boot	23
4.3 Linux kernel	24
4.4 STM32Cube	24
4.5 OP-TEE	25



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

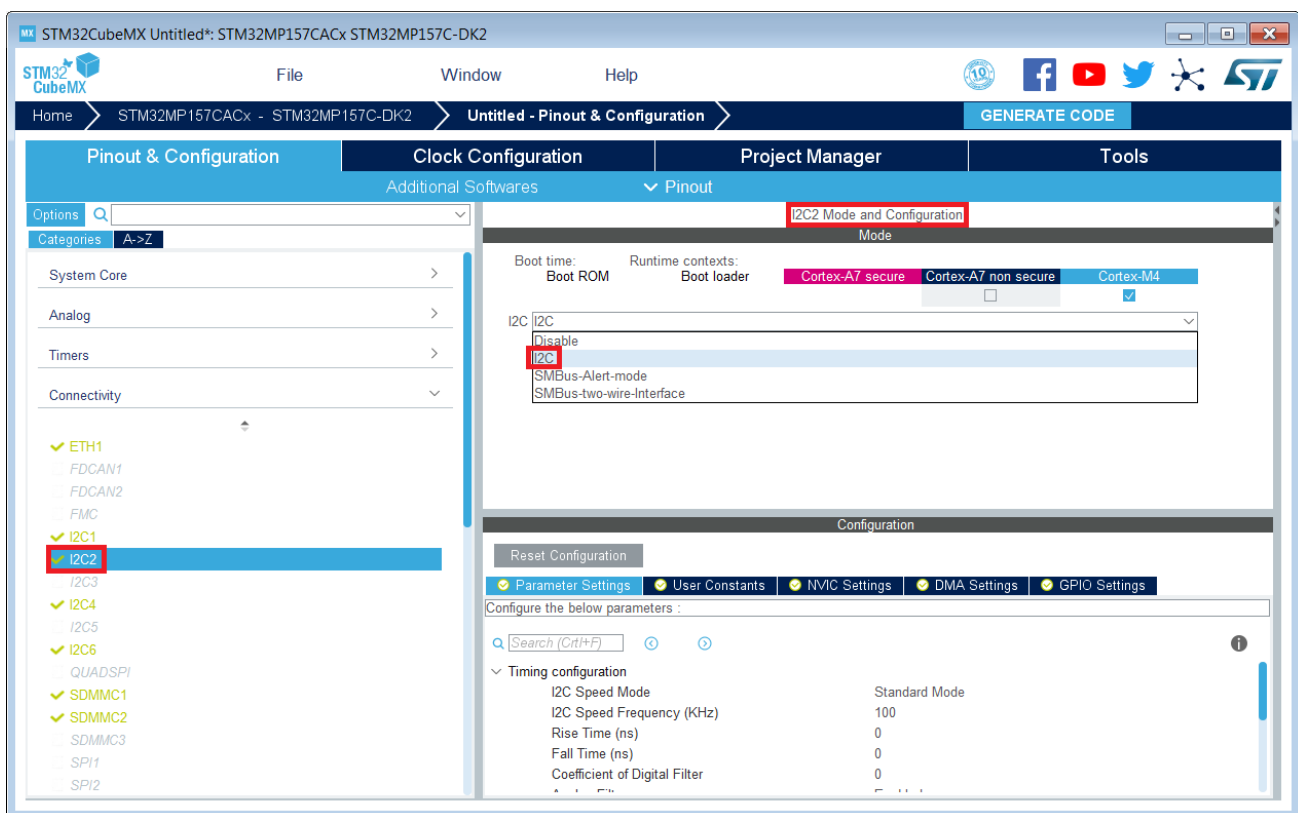
Beyond the peripherals assignment, explained in this article, it is also important to understand [How to configure system resources](#) (i.e clocks, regulator, gpio,...), shared between the Cortex-A7 and Cortex-M4 contexts



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/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";
};
```

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"*.

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: 17.11.2020 - 15:37 / Revision: 03.11.2020 - 13:18

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

Contents

1 Introduction	27
2 STM32Cube MP1 Package architecture	28
2.1 Level 0 (Drivers)	29
2.1.1 HAL drivers	29
2.1.1.1 HAL drivers overview	29
2.1.1.2 List of HAL drivers	29
2.1.2 LL drivers	30
2.1.2.1 Low Layer drivers overview	30
2.1.2.2 List of LL drivers	31
2.1.3 BSP drivers	31
2.1.3.1 BSP drivers overview	31



2.1.3.2 List of BSP drivers	31
2.2 Level 1 (Middlewares)	32
2.2.1 OpenAMP	32
2.2.2 FreeRTOS	32
2.3 Level 2 (Boards demonstrations)	33
2.4 Utilities	33
2.5 CMSIS	34
3 STM32Cube MP1 Package versus legacy STM32Cube MCU Package	37
4 References	38



1 Introduction

This article introduces **STM32Cube MP1 Package** architecture based on the Arm[®]Cortex[®]-M processor (e.g. Arm Cortex-M4)

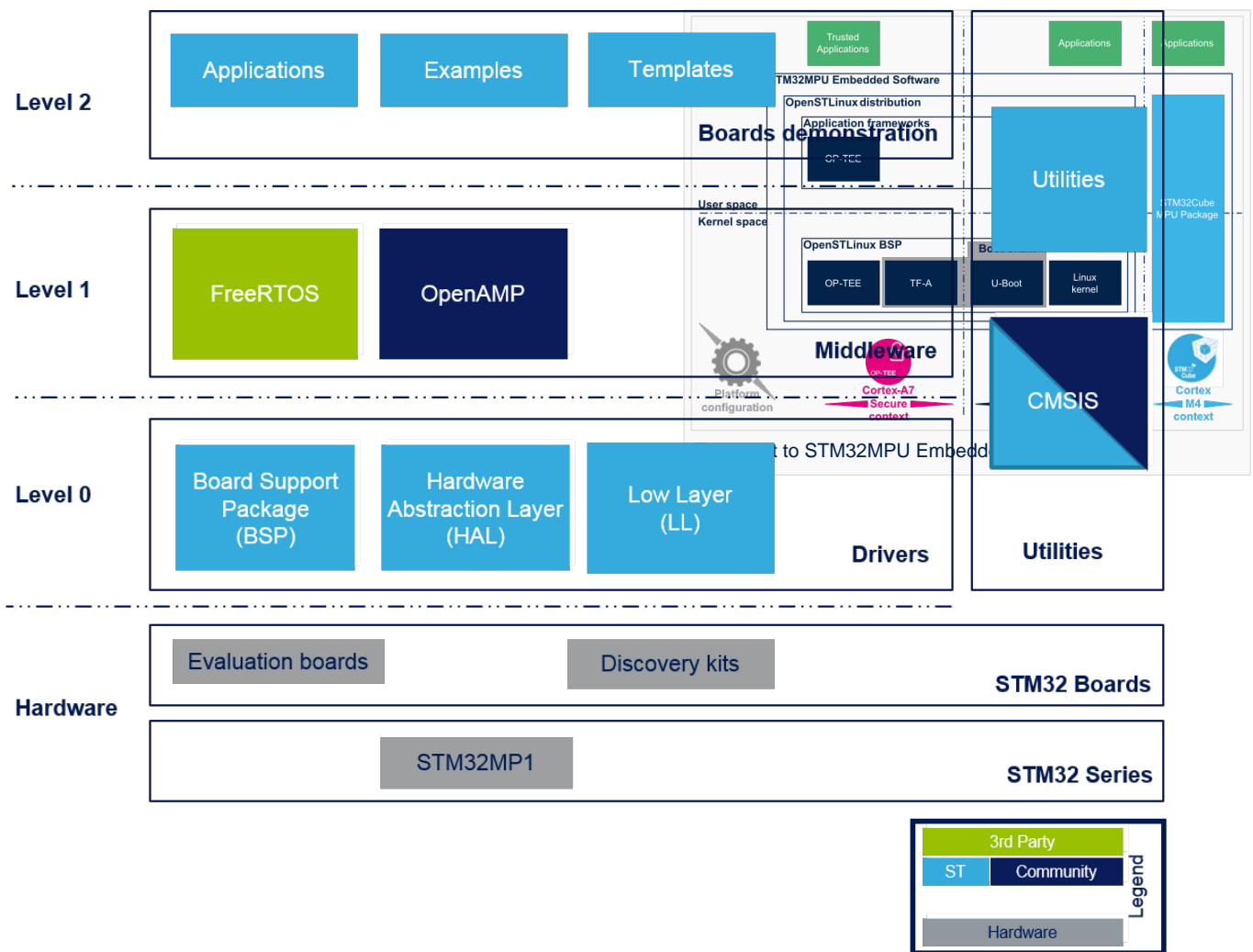
- Please refer to [STM32Cube MP1 Package](#) article to get started.



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

Legend:

(HAL drivers added since ecosystem release \geq v2.1.0 )

— stm32mp1xx_hal_adc.c ADC HAL Driver



stm32mplxx_hal_adc_ex.c	ADC Extended HAL Driver
stm32mplxx_hal_cec.c	CEC HAL Driver
stm32mplxx_hal_cortex.c	CORTEX HAL Driver
stm32mplxx_hal_crc.c	CRC HAL Driver
stm32mplxx_hal_crc_ex.c	CRC Extended HAL Driver
stm32mplxx_hal_cryp.c	CRYP HAL Driver
stm32mplxx_hal_cryp_ex.c	CRYP Extended HAL Driver
stm32mplxx_hal_dac.c	DAC HAL Driver
stm32mplxx_hal_dac_ex.c	DAC Extended HAL Driver
stm32mplxx_hal_dcmi.c	DCMI HAL Driver
stm32mplxx_hal_ddsdm.c	DFSDM HAL Driver
stm32mplxx_hal_ddsdm_ex.c	DFSDM Extended HAL Driver
stm32mplxx_hal_dma.c	DMA HAL Driver
stm32mplxx_hal_dma_ex.c	DMA Extended HAL Driver
stm32mplxx_hal_exti.c	EXTI HAL Driver
stm32mplxx_hal_fdcan.c	FDCAN HAL Driver
stm32mplxx_hal_gpio.c	GPIO HAL Driver
stm32mplxx_hal_gpio_ex.c	GPIO Extended HAL Driver
stm32mplxx_hal_hash.c	HASH HAL Driver
stm32mplxx_hal_hash_ex.c	HASH Extended HAL Driver
stm32mplxx_hal_hsem.c	HSEM HAL Driver
stm32mplxx_hal_i2c.c	I2C HAL Driver
stm32mplxx_hal_i2c_ex.c	I2C Extended HAL Driver
stm32mplxx_hal_ipcc.c	IPCC HAL Driver
stm32mplxx_hal_lptim.c	LPTIM HAL Driver
stm32mplxx_hal_mdios.c	MDIOS HAL Driver
stm32mplxx_hal_mdma.c	MDMA HAL Driver
stm32mplxx_hal_pwr.c	PWR HAL Driver
stm32mplxx_hal_pwr_ex.c	PWR Extended HAL Driver
stm32mplxx_hal_qspi.c	QSPI HAL Driver
stm32mplxx_hal_rcc.c	RCC HAL Driver
stm32mplxx_hal_rcc_ex.c	RCC Extended HAL Driver
stm32mplxx_hal_rng.c	RNG HAL Driver
stm32mplxx_hal_rtc.c	RTC HAL Driver
stm32mplxx_hal_rtc_ex.c	RTC Extended HAL Driver
stm32mplxx_hal_sai.c	SAI HAL Driver
stm32mplxx_hal_sai_ex.c	SAI Extended HAL Driver
stm32mplxx_hal_sd.c	SD HAL Driver
stm32mplxx_hal_smartcard.c	SMARTCARD HAL Driver
stm32mplxx_hal_smartcard_ex.c	SMARTCARD Extended HAL Driver
stm32mplxx_hal_smbus.c	SMBUS HAL Driver
stm32mplxx_hal_spdifrx.c	SPDIFRX HAL Driver
stm32mplxx_hal_spi.c	SPI HAL Driver
stm32mplxx_hal_spi_ex.c	SPI Extended HAL Driver
stm32mplxx_hal_sram.c	FMC HAL Driver (for PSRAM)
stm32mplxx_hal_tim.c	TIM HAL Driver
stm32mplxx_hal_tim_ex.c	TIM Extended HAL Driver
stm32mplxx_hal_uart.c	UART HAL Driver
stm32mplxx_hal_uart_ex.c	UART Extended HAL Driver
stm32mplxx_hal_usart.c	USART HAL Driver
stm32mplxx_hal_usart_ex.c	USART Extended HAL Driver
stm32mplxx_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[®]-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 Resource_manager_for_coprocessing

```



2.5 CMSIS

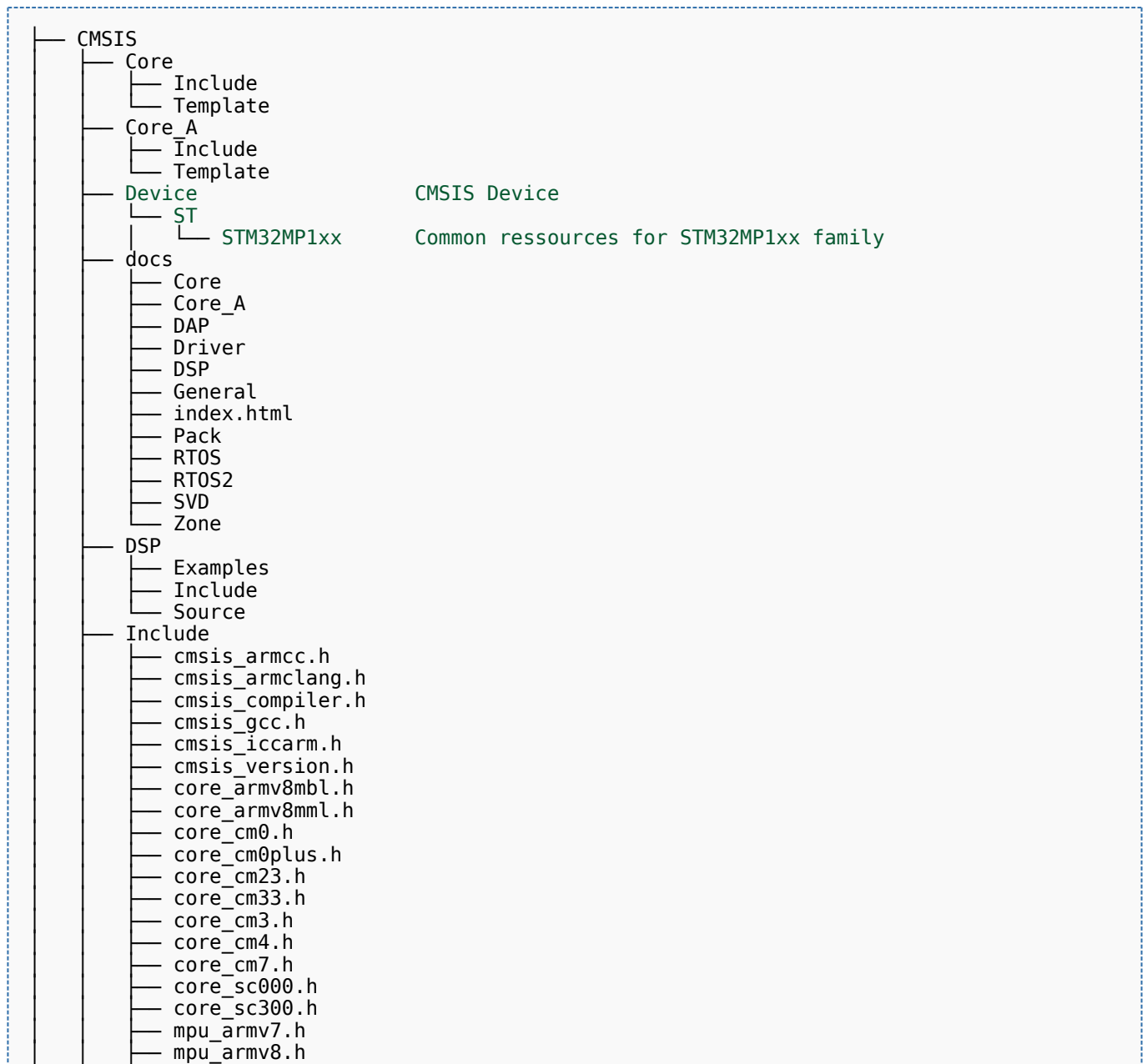
The **Cortex Microcontroller Software Interface Standard** (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series.

- 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:





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 offered are smaller in **STM32Cube MP1 Package** as we can take advantage of a rich OS like Linux[®] 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 processor between cortex A and cortex M (RPMmsg 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

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



Cortex®

Cortex Microcontroller Software Interface Standard

Hardware Abstraction Layer

Board support package

Linux® is a registered trademark of Linus Torvalds.

Device Tree

Low layer of STM32Cube

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

Universal Asynchronous Receiver/Transmitter

Direct Memory Access

Pulse Width Modulation

Analog-to-digital converter. The process of converting a sampled analog signal to a digital code that represents the amplitude of the original signal sample.

Consumer Electronics Control (HDMI standard)

Cyclic redundancy check calculation unit

Cryptographic processor

Digital-to-analog converter (Electronic circuit that converts a binary number into a continuously varying value.)

Digital Camera Memory Interface

Digital Filter for Sigma-Delta Modulator

External Interrupt

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.)

Hardware Semaphore

Inter-Processor Communication Controller

low-power timer (STM32 specific)

Reset and Clock Control

Random Number Generator

Real Time Clock

Serial Audio Interface (Mechanism used to transfer non-buffered audio data between processors and/or audio converters.)



Serial Peripheral Interface

Universal Synchronous/Asynchronous Receiver/Transmitter

System Configuration

Evaluation board

Discovery kit

Real Time Operating System

(Software)Integrated development/design/debugging environment

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.)

Microprocessor Unit

Operating System

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.

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](#).

NEW






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	 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	 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 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
Datasheets^[1]			
DS12505	STM32MP157C/F datasheet (secure)	DS12505.pdf	 v4.0
DS12504	STM32MP157A/D datasheet (basic)	DS12504.pdf	 v4.0
DS12503	STM32MP153C/F datasheet (secure)	DS12503.pdf	 v4.0
	STM32MP153A/D datasheet	DS125	



Reference	Name	Link	Version
Application notes			
DS12502	(basic)	02.pdf	 v4.0
DS12501	STM32MP151C/F datasheet (secure)	DS12501.pdf	 v4.0
DS12500	STM32MP151A/D datasheet (basic)	DS12500.pdf	 v4.0
DS12792	STPMIC1 datasheet	stpmic1.pdf	 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
 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



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	 v32.0
UM2237	STM32CubeProgrammer tool user manual	UM2237.pdf	 v12.0
UM2238	STM32 Trusted Package Creator tool user manual	UM2238.pdf	 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



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.

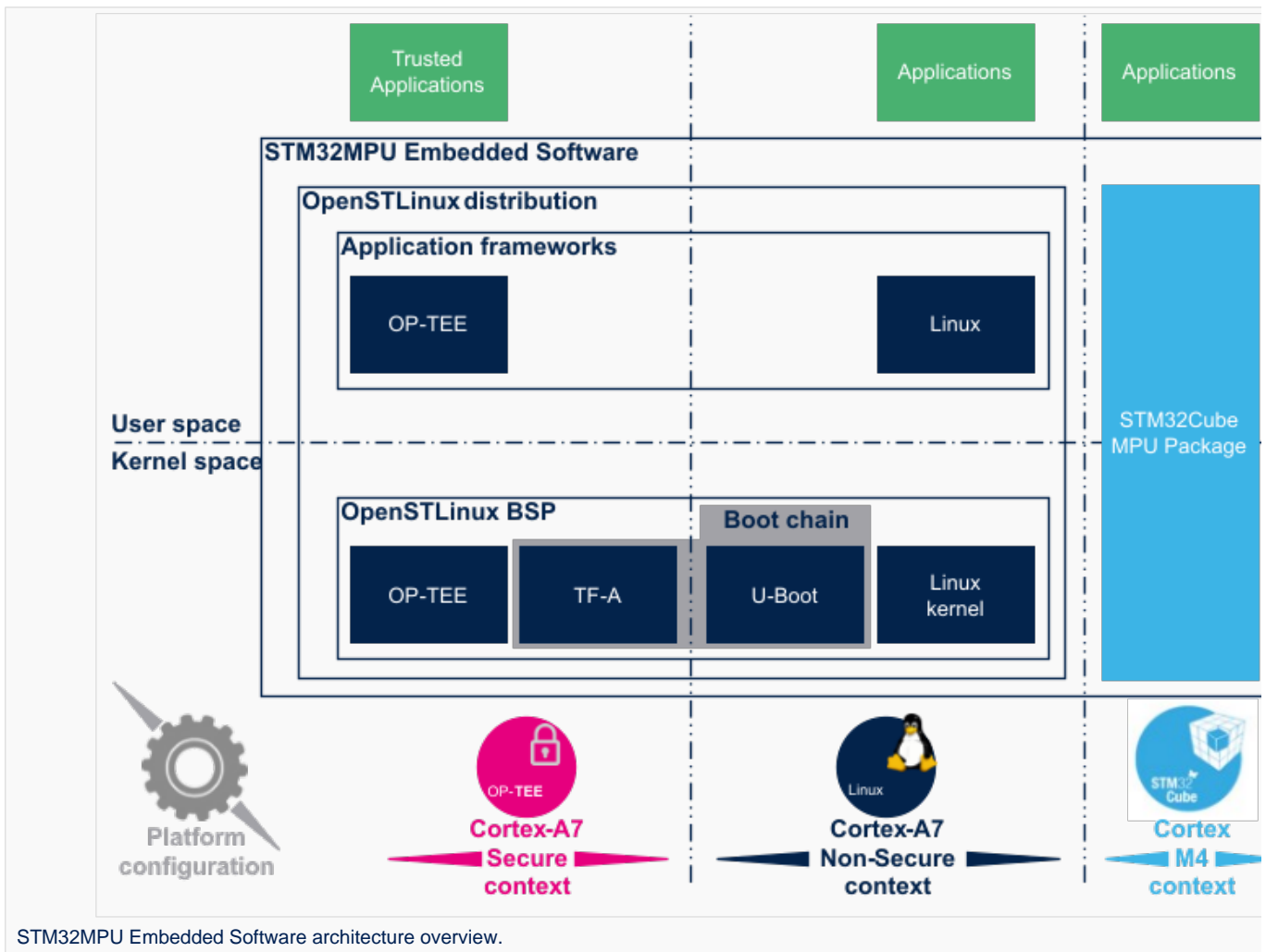


1 STM32MPU Embedded Software overview

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.





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^[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

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