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

STM32CubeMP1 architecture

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

Contents

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

Stable: 17.11.2020 - 15:37 / Revision: 05.11.2020 - 15:15

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

Contents

1 Introduction 4
2 STM32Cube MP1 Package architecture 5
2.1 Level 0 (Drivers) 6
2.1.1 HAL drivers 6
2.1.1.1 HAL drivers overview 6
2.1.1.2 List of HAL drivers 6
2.1.2 LL drivers 7
2.1.2.1 Low Layer drivers overview 7
2.1.2.2 List of LL drivers 8
2.1.3 BSP drivers 8
2.1.3.1 BSP drivers overview 8
2.1.3.2 List of BSP drivers 8
2.2 Level 1 (Middlewares) 9
2.2.1 OpenAMP 9
2.2.2 FreeRTOS 9
2.3 Level 2 (Boards demonstrations) 10
2.4 Utilities 10
2.5 CMSIS 11
3 STM32Cube MP1 Package versus legacy STM32Cube MCU Package 14
4 References 15

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

1 Introduction

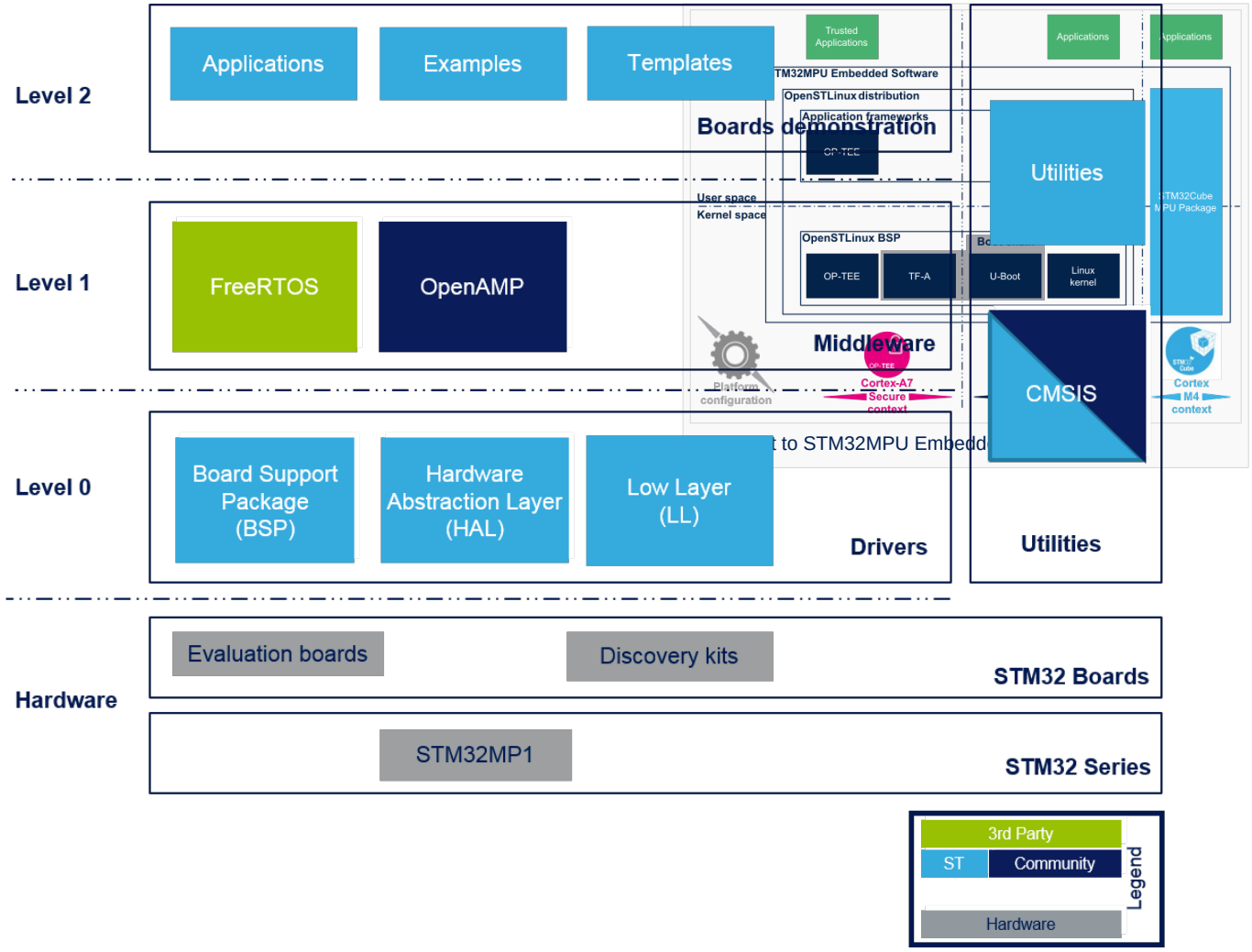
This article introduces **STM32Cube MP1 Package** architecture based on the Arm®Cortex®-M processor (e.g. Arm Cortex-M4)

- 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 ≥ v2.1.0 i )

├── 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_dfsdm.c         DFSDM HAL Driver
├── stm32mp1xx_hal_dfsdm_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®-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 fingerprint
- High portability
- Multithread management
- Pre-emptive scheduling
- Fast interrupt response
- Extensive inter-pocess 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
    └── ResourcesManager    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 ressources 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
│   ├── Core
│   │   ├── Include
│   │   └── Template
│   ├── Core_A
│   │   ├── Include
│   │   └── Template
│   ├── Device                  CMSIS Device
│   │   └── ST
│   │   │   └── STM32MP1xx      Common ressources for STM32MP1xx family
│   ├── docs
│   │   ├── Core
│   │   ├── Core_A
│   │   ├── DAP
│   │   ├── Driver
│   │   ├── DSP
│   │   ├── General
│   │   ├── index.html
│   │   ├── Pack
│   │   ├── RTOS
│   │   ├── RTOS2
│   │   ├── SVD
│   │   └── Zone
│   ├── DSP
│   │   ├── Examples
│   │   ├── Include
│   │   └── Source
│   ├── Include
│   │   ├── cmsis_armcc.h
│   │   ├── cmsis_armclang.h
│   │   ├── cmsis_compiler.h
│   │   ├── cmsis_gcc.h
│   │   ├── cmsis_iccarm.h
│   │   ├── cmsis_version.h
│   │   ├── core_armv8mbl.h
│   │   ├── core_armv8mml.h
│   │   ├── core_cm0.h
│   │   ├── core_cm0plus.h
│   │   ├── core_cm23.h
│   │   ├── core_cm33.h
│   │   ├── core_cm3.h
│   │   ├── core_cm4.h
│   │   ├── core_cm7.h
│   │   ├── core_sc000.h
│   │   ├── core_sc300.h
│   │   ├── mpu_armv7.h
│   │   ├── mpu_armv8.h
```

```
│   │   └── tz_context.h
│   ├── Lib
│   │   ├── ARM
│   │   └── GCC
│   ├── LICENSE.txt
│   ├── README.md
│   ├── RTOS
│   │   └── Template
│   └── RTOS2
│       ├── Include
│       ├── Source
│       └── Template
```

- **CMSIS Device** structure :

```
 Legend:
    (Header files added since ecosystem release ≥ v2.1.0 ℹ )

├── CMSIS
│   ├── Device                                  CMSIS Device
│   │   └── ST
│   │       └── STM32MP1xx                      Common ressources for STM32MP1xx family
│   │           ├── Include                     Device ressources definition
│   │           │   ├── stm32mp151axx_cm4.h
│   │           │   ├── stm32mp151axx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp151cxx_cm4.h
│   │           │   ├── stm32mp151cxx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp151dxx_cm4.h             --> for 800MHz
│   │           │   ├── stm32mp151dxx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp151fxx_cm4.h             --> for 800MHz
│   │           │   ├── stm32mp151fxx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp153axx_cm4.h
│   │           │   ├── stm32mp153axx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp153cxx_cm4.h
│   │           │   ├── stm32mp153cxx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp153dxx_cm4.h             --> for 800MHz
│   │           │   ├── stm32mp153dxx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp153fxx_cm4.h             --> for 800MHz
│   │           │   ├── stm32mp153fxx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp157axx_cm4.h
│   │           │   ├── stm32mp157axx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp157cxx_cm4.h
│   │           │   ├── stm32mp157cxx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp157dxx_cm4.h             --> for 800MHz
│   │           │   ├── stm32mp157dxx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp157fxx_cm4.h             --> for 800MHz
│   │           │   ├── stm32mp157fxx_ca7.h             --> Added "as example"
│   │           │   ├── stm32mp1xx.h
│   │           │   └── system_stm32mp1xx.h
│   │           ├── Release_Notes.html
│   │           └── Source
│   │               └── Templates
│   │                   ├── arm                                 linker files and startup
files (memory mapping)
│   │                   │   ├── linker
│   │                   │   │   └── stm32mp15xx_m4.sct
│   │                   │   ├── startup_stm32mp15xx.s
│   │                   ├── gcc                                 linker files and startup
files (memory mapping)
│   │                   │   ├── linker
│   │                   │   │   └── stm32mp15xx_m4.ld
│   │                   │   ├── startup_stm32mp15xx.s
│   │                   │   ├── startup_stm32mp151a_cm4.s      --> Added to fit with the
resources available
```

```
|   |   |   |   |   |   |   ├── startup_stm32mp151c_cm4.s      --> Added to fit with the
resources available
|   |   |   |   |   |   |   ├── startup_stm32mp153a_cm4.s      --> Added to fit with the
resources available
|   |   |   |   |   |   |   ├── startup_stm32mp153c_cm4.s      --> Added to fit with the
resources available
|   |   |   |   |   |   |   ├── startup_stm32mp157a_cm4.s      --> Added to fit with the
resources available
|   |   |   |   |   |   |   ├── startup_stm32mp157c_cm4.s      --> Added to fit with the
resources available
|   |   |   |   |   |   ├── iar                                 linker files and startup
files (memory mapping)
|   |   |   |   |   |   |   ├── linker
|   |   |   |   |   |   |   |   └── stm32mp15xx_sram.icf
|   |   |   |   |   |   |   ├── startup_stm32mp15xx.s
|   |   |   |   |   |   |   └── system_stm32mp1xx.c
```

Information

Notes:

- Several **CMSIS devices** are provided for a same family (ex: stm32mp157cxx.h & stm32mp157axx.h are provided for stm32mp1 family). It is done to fit exactly the resources present in the STM32 Part Number (ex: stm32mp157a does not include CRYP peripheral).
- Usage of the right **CMSIS device** is done thanks to a preprocessor switch in IDE project settings (ex: STM32MP157Axx, or STM32MP157Cxx, or STM32MP157Dxx, or STM32MP157Fxx)

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

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® 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

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.)

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

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