UM2204

User manual

Getting started with STM32CubeH7 for STM32H7 Series

Introduction

STM32Cube is an STMicroelectronics original initiative to significantly improve developer productivity by reducing development effort, time and cost. STM32Cube covers the whole STM32 portfolio.

STM32Cube includes:

- A set of user-friendly software development tools to cover project development from the conception to the realization, among which:
  - STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
  - STM32CubeIDE, an all-in-one development tool with peripheral configuration, code generation, code compilation, and debug features
  - STM32CubeProgrammer (STM32CubeProg), a programming tool available in graphical and command-line versions
  - STM32CubeMonitor-Power (STM32CubeMonPwr), a monitoring tool to measure and help in the optimization of the power consumption of the MCU.
- – STM32Cube MCU & MPU Packages, comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeH7 for the STM32H7 Series), which include:
    - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the STM32 portfolio
    - STM32Cube low-layer APIs, ensuring the best performance and footprints with a high degree of user control over the HW
    - A consistent set of middleware components such as RTOS, USB, TCP/IP, and Graphics
    - All embedded software utilities with full sets of peripheral and applicative examples
  - STM32Cube Expansion Packages, which contain embedded software components that complement the functionalities of the STM32Cube MCU & MPU Packages with:
    - Middleware extensions and applicative layers
    - Examples running on some specific STMicroelectronics development boards

This user manual describes how to get started with the STM32CubeH7 MCU Package. Section 1 STM32CubeH7 main features describes the main features of the STM32CubeH7 MCU Package.

Section 2 STM32CubeH7 architecture overview and Section 3 STM32CubeH7 MCU Package overview provide an overview of the STM32CubeH7 architecture and MCU Package structure.

1 STM32CubeH7 main features

STM32CubeH7 MCU Package runs on STM32H7 32-bit microcontrollers based on Arm® Cortex®-M processors. The STM32H7 products come with different lines mainly single core lines based on Cortex®-M7 and dual core lines based on a Cortex®-M7 + Cortex®-M4 architecture.

STM32CubeH7 gathers together, in a single package, all the generic embedded software components required to develop an application on STM32H7 microcontrollers. This set of components is highly portable, not only within the STM32H7 Series but also to other STM32 Series.

STM32CubeH7 is fully compatible with STM32CubeMX code generator that allows the user to generate initialization code. The package includes low-layer (LL) and hardware abstraction layer (HAL) APIs that cover the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL and LL APIs are available in an open-source BSD license for user convenience. They are compliant with MISRA C®:2012 guidelines, and have been reviewed with a static analysis tool to eliminate possible run-time errors. Reports are available on demand.

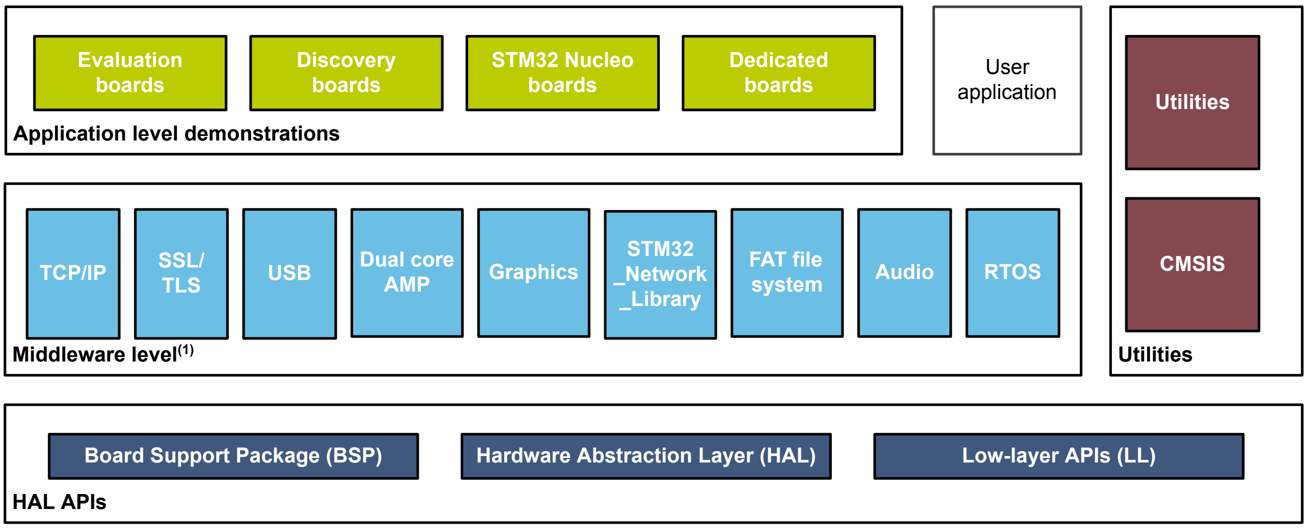
The STM32CubeH7 MCU Package also contains a set of middleware components with the corresponding examples. They come with very permissive license terms:

- Full USB Host and Device stack supporting many classes:
  – Host Classes: HID, MSC, CDC, Audio, MTP
  – Device Classes: HID, MSC, CDC, Audio, DFU
- Graphics:
  – STemWin, a professional graphical stack solution available in binary format and based on the emWin solution from ST's partner SEGGER
  – LibJPEG, an open source implementation on STM32 for JPEG images encoding and decoding
  – TouchGFX, a professional graphical stack solution from STMicroelectronics to create revolutionizing embedded graphical user interfaces (GUIs) with high-end graphics and maximum performance on energy efficient STM32 microcontrollers.
- Audio:
  – PDM2PCM library, offering a solution to decimate and filter out a pulse density modulated (PDM) stream from a digital microphone, in order to convert it to a pulse code modulated (PCM) signal output stream.
- CMSIS-RTOS implementation with FreeRTOS™ open source solution. This RTOS solution comes with dedicated communications primitives (Stream Buffers and Message Buffers), allowing to pass data from an interrupt service routine to a task, or from one core to another in STM32H7 dual core lines.
- OpenAMP , an AMP framework providing software components that enable development of software applications for asymmetric multiprocessing (Cortex®-M7 and Cortex-®-M4 in STM32H7 dual lines).
- FAT File system based on open source FatFS solution
- TCP/IP stack based on open source LwIP solution
- SSL/TLS secure layer based on open source mbedTLS
- Network Interface:
  – STM32_Network_Library provides APIs to access network services on STM32 devices. It supports several network adapters and protocols required by STM32Cube application using network services.

Several demonstrations implementing these middleware components are also provided in the STM32CubeH7 MCU Package.

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

Figure 1. **STM32CubeH7 firmware components**

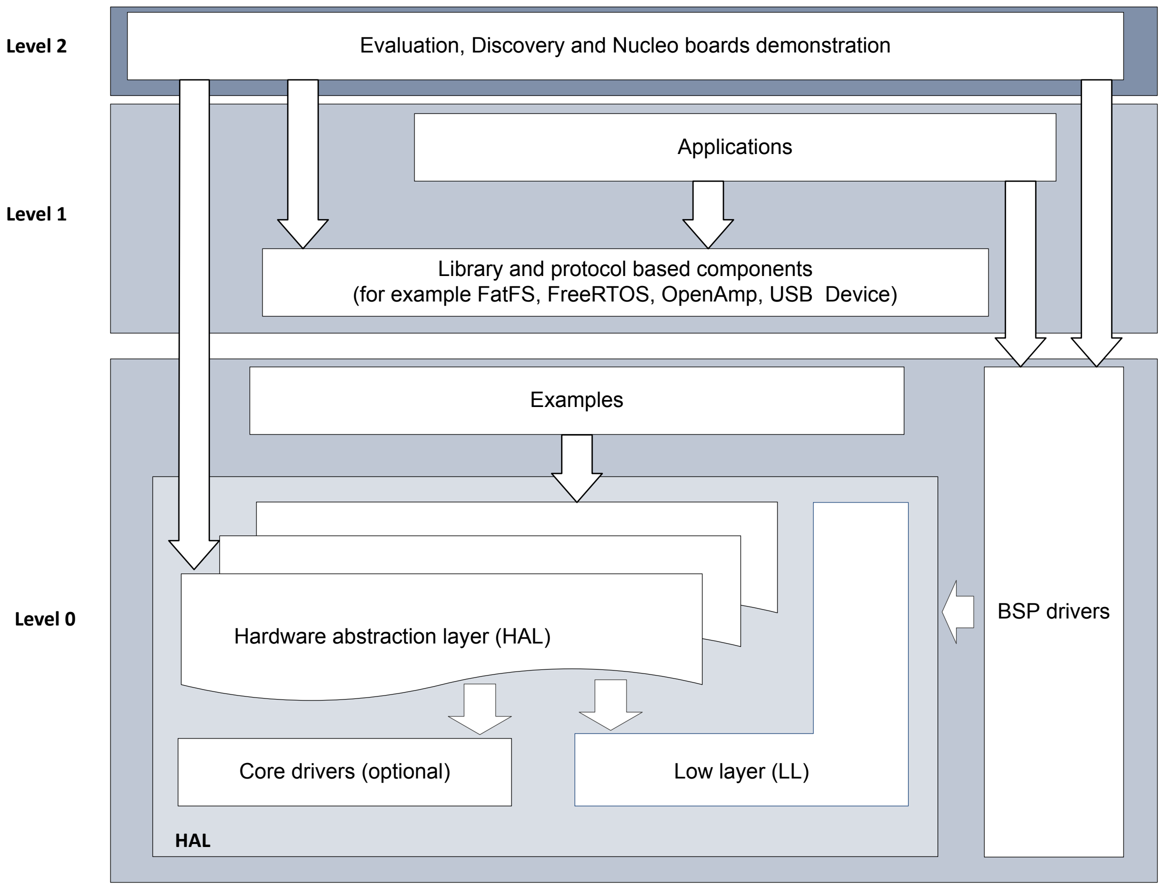


(1) The set of middleware components depends on the product Series.

2 STM32CubeH7 architecture overview

The STM32CubeH7 firmware solution is built around three independent levels that can easily interact with each other as described in Figure 3. STM32CubeH7 firmware architecture .

Figure 2. STM32CubeH7 firmware architecture


Level 2
Evaluation, Discovery and Nucleo boards demonstration
Level 1
Applications
Library and protocol based components
(for example FatFS, FreeRTOS, OpenAmp, USB Device)
Level 0
Examples
Hardware abstraction layer (HAL)
BSP drivers
Core drivers (optional)
Low layer (LL)
HAL

2.1 Level 0

This level is divided into three sub-layers:

- Board support package (BSP)
- Hardware abstraction layer (HAL)
- Low layer drivers
- Basic peripheral usage examples

2.1.1 Board support package (BSP)

This layer offers a set of APIs relative to the hardware components in the hardware boards (such as Audio codec, I/O expander, Touchscreen, SRAM driver or LCD drivers). It is composed of two parts:

- Component

  This is the driver related to the external device on the board and not related to the STM32, the component driver provides specific APIs to the BSP driver external components and can be ported to any board.

- BSP driver

  It allows linking the component driver to a specific board and provides a set of user-friendly APIs. The API naming rule is BSP_FUNCT_Action(): ex. BSP_LED_Init(),BSP_LED_On()

The BSP is based on a modular architecture allowing an easy porting on any hardware by just implementing the low level routines.

2.1.2 Hardware abstraction layer (HAL) and low-layer (LL)

The HAL layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi instance and function-oriented APIs which simplify the user application implementation by providing ready-to-use processes. As example, for the communication peripherals (I2S, UART…), it includes APIs allowing to initialize and configure the peripheral, manage data transfer based on polling, interrupt or DMA process, and handle communication errors that may raise during communication. 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 and customized functions for a specific family or a specific part number
- The low-layer APIs provide low-level APIs at register level, with better optimization but less portability. They require a deep knowledge of MCU and peripheral specifications. The LL drivers are designed to offer a fast light-weight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, LL APIs are not provided for peripherals where optimized access is not a key feature, or for those requiring heavy software configuration and/or complex upper-level stack. The LL drivers feature:
  - A set of functions to initialize peripheral main features according to the parameters specified in data structures
  - A set of functions used to fill initialization data structures with the reset values corresponding to each field
  - Function for peripheral de-initialization (peripheral registers restored to their default values)
  - A set of inline functions for direct and atomic register access
  - Full independence from HAL and capability to be used in standalone mode (without HAL drivers)
  - Full coverage of the supported peripheral features.
- Dual Core implementation:
  - The same HAL/LL drivers support both the single and dual core STM32H7 lines
    - In the STM32H7 dual core devices all peripherals can be accessed in the same way by the two cores (Cortex®-M7 and Cortex®-M4). It means there is no peripherals split or default allocation between Cortex®-M7 and Cortex®-M4. For this reason the same peripheral HAL and LL drivers are shared between the two cores.
    - Furthermore, some peripherals (mainly: RCC, GPIO, FLASH, PWR, HSEM..) have additional dual core specific features:
      - “DUAL_CORE” define is used to delimit code (defines , functions, macros..) available only in dual core line.
      - “CORE_CM4” define is used to delimit code where we a specific configuration/code portion for Cortex®-M4 core
      - “CORE_CM7” define is used to delimit code where we a specific configuration/code portion for Cortex®-M7 core on a dual core line.

2.1.3 Basic peripheral usage examples

This layer contains the examples of the basic operation of the STM32H7 peripherals using either the HAL APIs as well as the BSP resources.

2.2 Level 1

This level is divided into two sub-layers:

2.2.1 Middleware components

Middleware components are a set of libraries covering USB Host and Device Libraries, STemWin, TouchGFX, LibJPEG, FreeRTOS™, OpenAMP, FatFS, LwIP, mbedTLS, and PDM2PCM audio library. Horizontal interactions between the components of this layer are performed directly by calling the feature APIs while the vertical interaction with the low level drivers is done through specific callbacks and static macros implemented in the library system call interface. As example, the FatFs implements the disk I/O driver to access microSD drive or the USB Mass Storage Class.

The main features of each middleware component are as follows:

- **USB host and device libraries**
  - Several USB classes supported (mass-storage, HID, CDC, DFU, AUDIO, MTP)
  - Support of multipacket transfer features: allows sending big amounts of data without splitting them into max packet size transfers.
  - Use of configuration files to change the core and the library configuration without changing the library code (read-only).
  - 32-bit aligned data structures to handle DMA-based transfer in High-speed modes.
  - Support of multi USB OTG core instances from user level through configuration file (that allows an operation with more than one USB host/device peripheral).
  - RTOS and Standalone operation
  - The link with low-level driver through an abstraction layer using the configuration file to avoid any dependency between the Library and the low-level drivers.
- **STemWin Graphical stack**
  - Professional grade solution for GUI development based on Segger’s emWin solution
  - Optimized display drivers
  - Software tools for code generation and bitmap editing (STemWin Builder…)
- **LibJPEG**
  - Open source standard
  - C implementation for JPEG image encoding and decoding.
- **TouchGFX Graphical stack**
  - Professional graphical stack solution from STMicroelectronics
  - Allotting to create revolutionizing embedded graphical user interfaces (GUIs) with high-end graphics and maximum performance on energy efficient STM32 microcontrollers
- **FreeRTOS**
  - Open source standard
  - CMSIS compatibility layer
  - Tickless operation during low-power mode
  - Integration with all STM32Cube middleware modules
- **OpenAmp**
  - AMP framework providing for asymmetric multiprocessing (Cortex®-M7 and Cortex®-M4 in STM32H7 dual lines)
- **FAT File system**
  - FATFS FAT open source library
  - Long file name support
  - Dynamic multi-drive support
  - RTOS and standalone operation
  - Examples with microSD and USB host mass-storage class
- **LwIP TCP/IP stack**
  - Open source standard
  - RTOS and standalone operation

- **PDM2PCM audio library**
  - Sampling rate of 16 kHz with a 16-bit resolution
  - Various decimation factors to adapt to various PDM clocks
  - Configurable digital microphone gain with 1 dB step in the range of -12 dB to +51 dB
- **STM32_Network_Library**
  - APIs for accessing network services on STM32 devices
  - Several network adapters and protocols

2.2.2 Examples based on the middleware components

Each middleware component comes with one or more examples (called also Applications) showing how to use it. Integration examples that use several middleware components are provided as well.

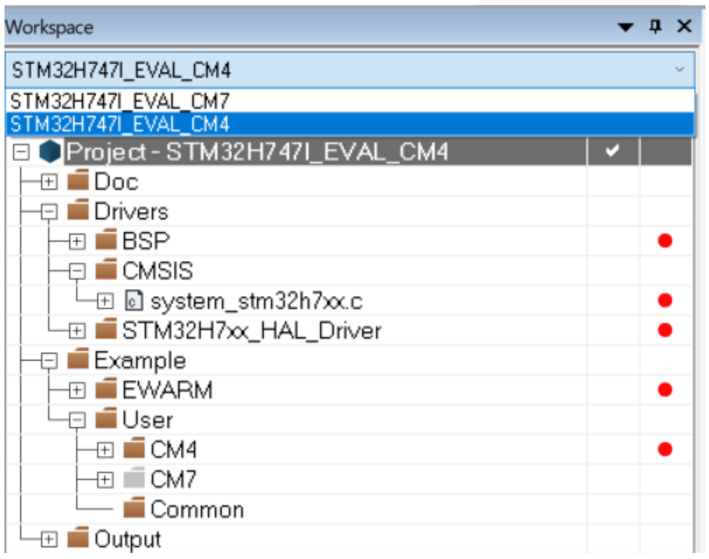
2.3 Level 2

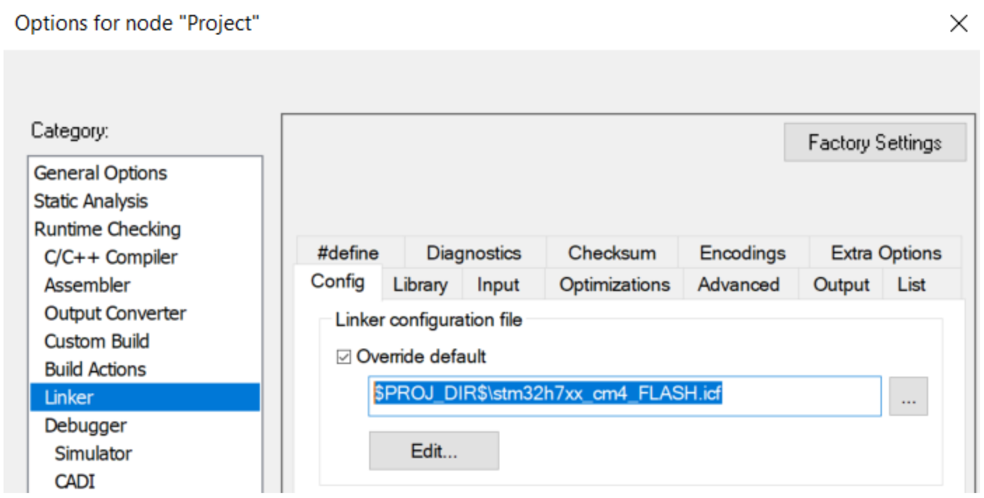
This level is composed of a single layer which is a global real-time and graphical demonstration based on the middleware service layer, the low level abstraction layer and the basic peripheral usage applications for board-based features.

Dual core examples and applications follow a dedicated architecture:

- In dual core lines, only one project (one workspace) per example/application is provided. This is to be compatible with legacy (single core lines) offer.
- Two target projects configuration per workspace (one per core) named STM32YYxx_CM7 and STM32YYxx_CM4.
- Each target configuration has its own option settings: target device, linker options, RO, RW zones, preprocessor symbols (CORE_CMx , CORE_CMy) so that user code can be compiled, linked and programmed separately for each core. The compilation results in 2 binaries: CM7 binary and CM4 binary.

Figure 3. **Dual core project architecture**





Linker configuration file

☑ Override default

$PROJ_DIR$\stm32h7xx_cm4_FLASH.icf

Figure 4. **Dual core project linker file**

3 STM32CubeH7 MCU Package overview

3.1 Supported STM32H7 devices and hardware

STM32Cube offers a highly portable Hardware Abstraction Layer (HAL) built around a generic and modular architecture. It allows the upper layers, the middleware and application, to implement its functions without knowing, in-depth, the MCU used. This improves the library code re-usability and guarantees an easy portability from one device to another.

The STM32CubeH7 offers a full support for all the STM32H7 devices. The user only needs to define the right macro in the *stm32h7xx.h* file.

Table 1. Macros for STM32H7 Series lists which macro to define depending on the used STM32H7 Series device. This macro can also be defined in the compiler preprocessor.

Table 1. Macros for STM32H7 Series

Macro defined in stm32h7xx.h	STM32H7 part numbers
STM32H742xx	STM32H742VI, STM32H742ZI, STM32H742II, STM32H742BI, STM32H742XI
STM32H743xx	STM32H743VI, STM32H743ZI, STM32H743II, STM32H743BI, STM32H743XI
STM32H745xx(1)	STM32H745ZI, STM32H745II, STM32H745BI, STM32H745XI
STM32H747xx(1)	STM32H747ZI, STM32H747AI, STM32H747II, STM32H747BI, STM32H747XI
STM32H750xx	STM32H750VB, STM32H750IB, STM32H750XB
STM32H753xx	STM32H753VI, STM32H753ZI, STM32H753II, STM32H753BI, STM32H753XI
STM32H755xx(1)	STM32H755ZI, STM32H755II, STM32H755BI, STM32H755XI
STM32H757xx(1)	STM32H757ZI, STM32H757AI, STM32H757II, STM32H757BI, STM32H757XI
STM32H7B0xx	STM32H7B0ZB, STM32H7B0VB, STM32H7B0RB, STM32H7B0IB, STM32H7B0AB
STM32H7A3xx	STM32H7B3AI, STM32H7A3ZI, STM32H7A3ZG, STM32H7A3VI, STM32H7A3VG, STM32H7A3RI, STM32H7A3RG, STM32H7A3QI, STM32H7A3NI, STM32H7A3NG, STM32H7A3LI, STM32H7A3II, STM32H7A3IG, STM32H7A3AI, STM32H7A3AG
STM32H7B3xx	STM32H753BI, STM32H7B3ZI, STM32H7B3VI, STM32H7B3RI, STM32H7B3QI, STM32H7B3NI, STM32H7B3LI, STM32H7B3II

1. *STM32H745xx, STM32H747xx, STM32H755xx, and STM32H757xx are dual core devices. When using these devices, either CORE_CM7 or CORE_CM4 (depending of the target core configuration) shall be added into the compiler preprocessor.*

The STM32CubeH7 features a rich set of examples and demonstrations at all levels making it easy to understand and use any HAL driver and/or middleware components. These examples can be run on any of the STMicroelectronics boards as listed in Table 2. Evaluation, Discovery and Nucleo boards for STM32H7 Series :

Table 2. Evaluation, Discovery and Nucleo boards for STM32H7 Series

Board	Supported STM32H7 lines
STM32H743I-EVAL	STM32H743xx, STM32H753xx and STM32H750xx (1)
NUCLEO-H743ZI	
NUCLEO-H745ZI-Q	STM32H745xx and STM32H755xx
STM32H745I-DISCO	STM32H745xx and STM32H755xx
STM32H747I-DISCO	STM32H747xx and STM32H757xx
STM32H747I-EVAL	STM32H747xx and STM32H757xx
STM32H750B-DK	STM32H750xx

Board	Supported STM32H7 lines
STM32H7B3I-EVAL	STM32H7B3xxQ and STM32H7B0xxQ(2)
STM32H7B3I-DK	
NUCLEO-H7A3ZI-Q	STM32H7A3xxQ

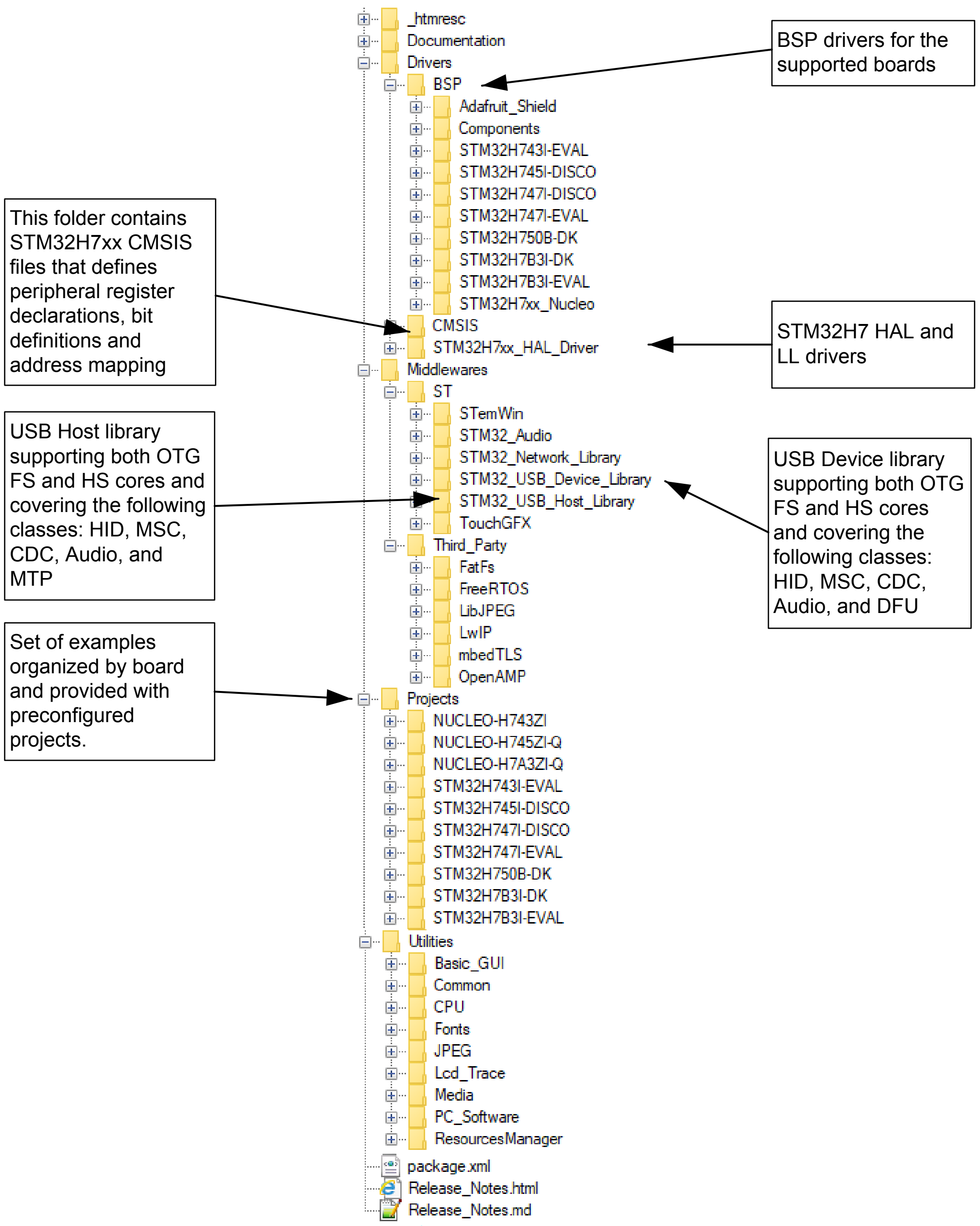
1. *The STM32H750xx belong to STM32H750 Value line. They feature only 128 Kbytes of internal Flash memory. These devices are intended for code execution from external memories. Dedicated applications are available under Projects \STM32H743I-EVAL\Applications\ExtMem_CodeExecution*
2. *STM32H7B0xx devices belong to STM32H7B0 Value Line. They feature only 128 Kbytes of internal Flash memory. They are intended for code execution from external memories. Dedicated applications are available under Projects \STM32H7B3I-DK\Applications\ExtMem_CodeExecution.*

The STM32CubeH7 MCU Package can run on any compatible hardware. Simply update the BSP drivers to port the provided examples on the user board if its hardware features are the same (LED, LCD display, pushbuttons).

3.2 MCU Package overview

The STM32CubeH7 firmware solution is provided in a single zip package with the structure shown in Figure 6. STM32CubeH7 MCU Package structure .

Figure 5. **STM32CubeH7 MCU Package structure**



For each board, a set of examples is provided with preconfigured projects for EWARM and MDK-ARM toolchains. Projects for SW4STM32 toolchain are provided for all boards except for STM32H7B3I-EVAL, STM32H7B3I-DK and NUCLEO-H7A3ZI-Q where projects for STM32CubeIDE are offered.

Figure 8. STM32CubeH7 example overview shows the project structure for the STM32H743I-EVAL board. The structure is identical for other boards.

The examples are classified depending on the STM32Cube level they apply to, and are named as follows:

- Examples in level 0 are called Examples. They use the HAL drivers without any middleware component.
- Examples in level 1 are called Applications, that provide typical use cases of each middleware component
- Examples in level 2 are called Demonstration, that implement all the HAL, BSP and middleware components

Template projects (HAL and LL) are provided to allow quickly build any firmware application on a given board.

For single core boards (STM32H43I-EVAl, STM32H7B3I-EVAL, STM32H7B3I-DK, NUCLEO-H7A3ZI-Q and NUCLEO-H743ZI): one HAL and one LL template project.

For dual core boards (NUCLEO-H745ZI-Q, STM32H745I-DISCO, STM32H747I-DISCO and STM32H747I-EVAL):

- One LL template project
- Four HAL template projects:
  - BootCM4_CM7:
    - Cortex®-M7 and Cortex®-M4 running from the Flash memory (each from a bank)
    - System configuration performed by the Cortex®-M7
    - Cortex®-M4 goes to STOP after boot, then woken-up by Cortex®-M7 using a hardware semaphore
  - BootCM7_CM4Gated:
    - Cortex®-M4 boot is gated using Flash memory option bytes
    - Cortex®-M7 and Cortex®-M4 running from the Flash memory (each from a bank)
    - Cortex®-M7 boots , performs the system configuration then enable the Cortex®-M4 boot using RCC
  - BootCM4_CM7Gated:
    - Cortex®-M7 boot is gated using Flash memory option bytes
    - Cortex®-M7 and Cortex®-M4 running from the Flash memory (each from a bank)
    - Cortex®-M4 boots , performs the system configuration then enables the Cortex®-M7 boot using RCC
  - BootCM7_CM4Gated_RAM:
    - Cortex®-M4 boot is gated using Flash memory option bytes
    - Cortex®-M7 running from the Flash memory (Bank1), Cortex®-M4 running from the D2 domain SRAM
    - Cortex®-M7 boots:
    - Performs the system configuration
    - Loads the Cortex®-M4 code into the D2 SRAM
    - Changes the Cortex®-M4 boot address then enable Cortex®-M4 boot (using the RCC)

For STM32H750B-DK Value line board the template is composed of two subtemplate projects:

- ExtMem_Boot: reference boot code executing from internal Flash memory, enabling to configure external memories, then jumping to the user application located in an external memory. Two use cases are possible, XiP and BootROM:
  - XiP: this use case is intended for eXecution in Place from external Flash memory (Quad-SPI). The user application code shall be linked with the target execution memory address (external Quad-SPI Flash memory)
  - BootROM: this use case is intended to demonstrate how to boot from internal Flash memory, configure the external SDRAM, copy user application binary from the SDMMC Flash memory or from Quad-SPI Flash memory to the external SDRAM then jump to the user application. In this case, the user application code shall be linked with the target execution memory address (external SDRAM)
- Template_Project: typical template with execution from external memory. Different configurations are available depending on the possibilities offered by the external memory boot:
  - XiP from Quad-SPI Flash, data stored in internal SRAM
  - XiP from Quad-SPI Flash, data stored in external SDRAM
  - BootROM: execution from external SDRAM, data stored in internal SRAM

For STM32H7B0xx Value line devices based on STM32H7B3I-DK board, ExtMem_CodeExecution applications are provided with two sub-applications:

- ExtMem_Boot: reference boot code executing from internal Flash memory, enabling to configure external memories, then jumping to the user application located in an external memory. Two use cases are possible, XiP and BootROM:
  – XiP: this use case is intended for eXecution in Place from external Flash memory (Octo-SPI). The user application code shall be linked to the target execution memory address (external Octo-SPI NOR Flash memory).
  – BootROM: this use case is intended to demonstrate how to boot from internal Flash memory, configure the external SDRAM, copy user application binary from the SDMMC Flash memory or from Octo-SPI Flash memory to the external SDRAM, and then jump to the user application. The user application code shall be linked to the target execution memory address (external SDRAM)
- ExtMem_Application: two sub-projects are provided (LedToggling and FreeRTOS) with execution from external memory.
  – XiP from Octo-SPI Flash memory, data stored in external SDRAM
  – XiP from Octo-SPI Flash memory, data stored in internal SRAM
  – BootROM: execution from external SDRAM, data stored in internal SRAM

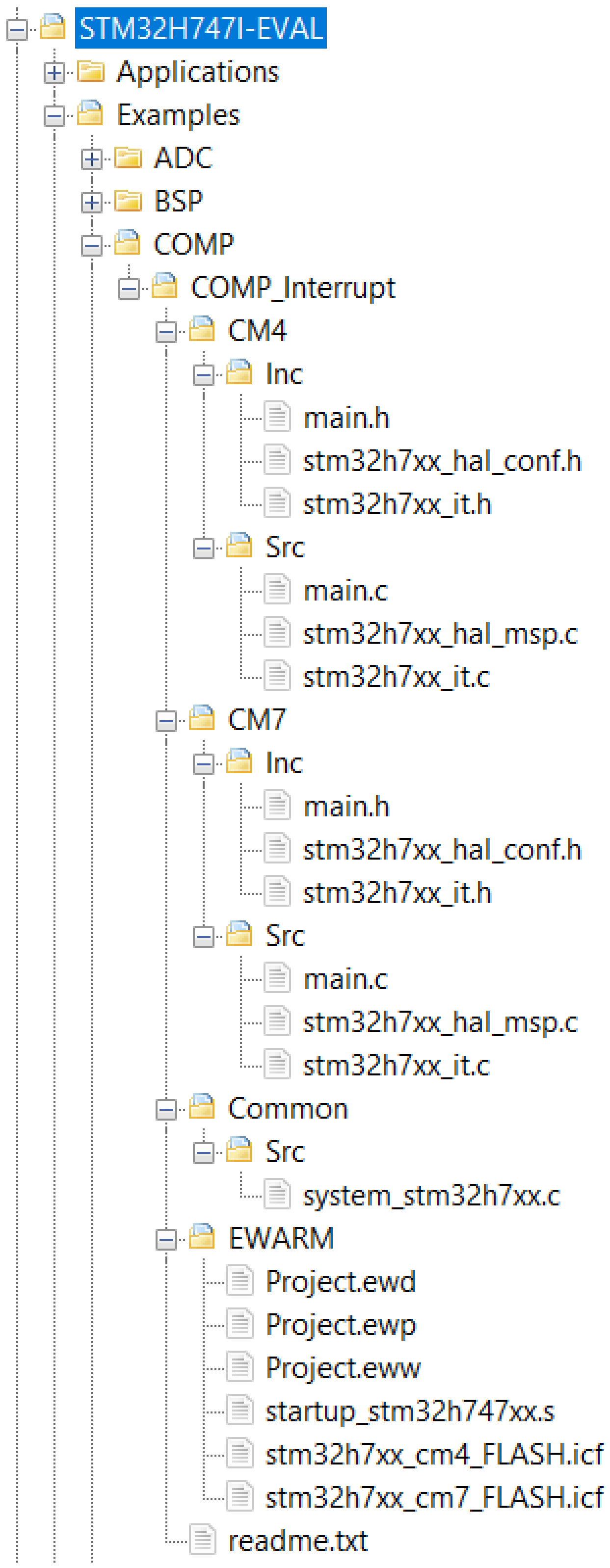
Single core examples have the same structure:

- \Inc folder that contains all header files
- \Src folder for the sources code
- \EWARM, \MDK-ARM and \SW4STM32 or \STM32CubeIDE folders contain the preconfigured project for each toolchain.
- readme.txt describing the example behavior and the environment required to make it work

All dual core examples have the same structure:

- Two separate folders CM4 and CM7 respectively for Cortex®-M4 and Cortex®-M7
- Each folder (CM4 and CM7) provides:
  – \Inc folder that contains all header files for Cortex®-M4/M7
  – \Src folder for the sources code files for Cortex®-M4/M7
- A \common folder with \Inc and \Src containing the common header and source files for both cores.

- \EWARM, \MDK-ARM and \SW4STM32 folders contain the preconfigured project for each toolchain (both Cortex®-M4 and Cortex®-M7 target configuration)



```
STM32H747I-EVAL
  Applications
  Examples
    ADC
    BSP
    COMP
      COMP_Interrupt
        CM4
          Inc
            main.h
            stm32h7xx_hal_conf.h
            stm32h7xx_it.h
          Src
            main.c
            stm32h7xx_hal_msp.c
            stm32h7xx_it.c
        CM7
          Inc
            main.h
            stm32h7xx_hal_conf.h
            stm32h7xx_it.h
          Src
            main.c
            stm32h7xx_hal_msp.c
            stm32h7xx_it.c
        Common
          Src
            system_stm32h7xx.c
        EWARM
          Project.ewd
          Project.ewp
          Project.eww
          startup_stm32h747xx.s
          stm32h7xx_cm4_FLASH.icf
          stm32h7xx_cm7_FLASH.icf
        readme.txt
```

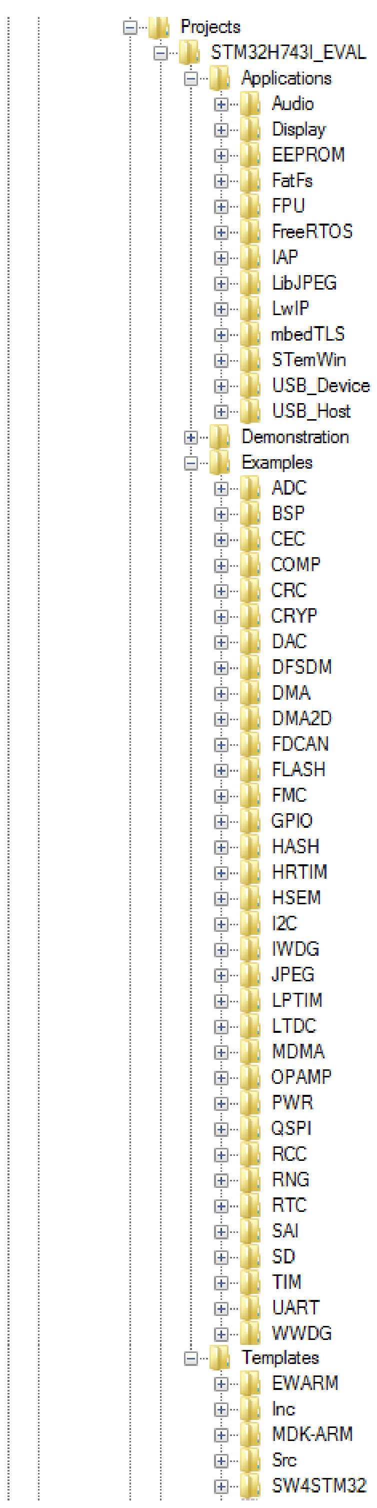
Figure 6. Dual core example structure

Section 3.2 provides the number of examples, applications and demonstrations available for each board.

Table 3. **Number of examples available for each board**

Board	Templates_LL	Templates	Examples	Examples_LL	Examples_MIX	Applications	Demonstration
NUCLEO-H743ZI	1	1	79	4	-	6	1
NUCLEO-H745ZI-Q	1	4	21	-	-	2	1
STM32H743I-EVAL	1	1	135	-	-	54	1
STM32H745I-DISCO	1	4	40	-	-	13	1
STM32H747I-DISCO	1	4	51	-	-	17	4
STM32H747I-EVAL	1	4	61	-	-	27	4
STM32H750B-DK	-	2	13	-	-	7	1
STM32H7B3I-EVAL	1	1	69	-	-	11	1
STM32H7B3I-DK	1	1	27	1	1	19	1
NUCLEO-H7A3ZI-Q	1	1	50	13	3	13	1

Figure 7. STM32CubeH7 example overview



```
Projects
  STM32H743I_EVAL
    Applications
      Audio
      Display
      EEPROM
      FatFs
      FPU
      FreeRTOS
      IAP
      LibJPEG
      LwIP
      mbedTLS
      STemWin
      USB_Device
      USB_Host
    Demonstration
    Examples
      ADC
      BSP
      CEC
      COMP
      CRC
      CRYP
      DAC
      DFSDM
      DMA
      DMA2D
      FDCAN
      FLASH
      FMC
      GPIO
      HASH
      HRTIM
      HSEM
      I2C
      IWDG
      JPEG
      LPTIM
      LTDC
      MDMA
      OPAMP
      PWR
      QSPI
      RCC
      RNG
      RTC
      SAI
      SD
      TIM
      UART
      WWDG
    Templates
      EWARM
      Inc
      MDK-ARM
      Src
      SW4STM32
```

4 Getting started with STM32CubeH7

4.1 Running your first example

This section explains how simple it is to run a first example with STM32CubeH7. It uses as an illustration the generation of a simple LED toggling example.

After downloading the STM32CubeH7 MCU package, unzip it into a directory of your choice, make sure not to modify the package structure shown in Figure 6. STM32CubeH7 MCU Package structure

1. Case of a single core project, STM32H743I-EVAL board:
   a. Browse to \\Projects\\STM32H743I-EVAL\\Examples.
   b. Open \\GPIO, then the \\GPIO_EXTI folder.
   c. Open the project with your preferred toolchain.
   d. Rebuild all files and load your image into target memory.
   e. Run the example: each time you press the **Tamper** push-button, the LED1 will toggle (for more details, refer to the example readme file).
2. Case of a dual core project, example STM32H747I-EVAL board:
   a. Browse to \\Projects\\STM32H747I-EVAL\\Examples.
   b. Open \\GPIO, then the \\GPIO_EXTI folder.
   c. Open the project with your preferred toolchain.
   d. For each target STM32H747I_EVAL_CM4 and STM32H747I_EVAL_CM7 (respectively for Cortex®-M7 and Cortex®-M4):
      i. Rebuild all files and load your image into target memory.
      ii. After loading the two images, reset the board in order to boot (Cortex®-M7) and CPU2 (Cortex®-M4) at once
      iii. Each time you press the **Tamper** push-button, :
         - LED1 toggles once (to indicated an EXTI interrupt for Cortex®-M7)
         - LED3 toggles once (to indicated an EXTI interrupt for Cortex®-M4) (for more details, refer to the example readme file)
3. Case of a Value line project running on STM32H750B-DK board:
   a. Browse to *Projects*\\STM32H750B-DK\\*Templates**ExtMem_Boot*.
   b. Open the ExtMem_Boot project with your preferred toolchain.
   c. Rebuild all files and load your image into the target internal Flash memory.
   d. Browse to \\Projects\\STM32H750B-DK\\Examples.
   e. Open \\GPIO, then the \\GPIO_IOToggle folder.
   f. Open the project with your preferred toolchain (keep the default configuration XIP_QSPI_InternalSRAM).
   g. Rebuild all files and load your image into the external Quad-SPI Flash memory.
   h. Run the example: LED1 will toggle infinitely (for more details, refer to the example readme file).
4. Case of a Value line project running on STM32H7B3I-DK board:
   a. Browse to *Projects\STM32H7B3I-DK\Applications\ExtMem_CodeExecution\ExtMem_Boot*.
   b. Open the ExtMem_Boot project with your preferred toolchain.
   c. Rebuild all files and load your image into target internal Flash memory.
   d. Browse to *Projects\STM32H7B3I-DK\Applications\ExtMem_CodeExecution\ExtMem_Application*.
   e. Open *LedToggling* folder.
   f. Open the project with your preferred toolchain (keep the default configuration XIP_OSPI_InternalSRAM).
   g. Rebuild all files and load your image into the external Octo-SPI Flash memory.
   h. Run the example: LED2 will toggle infinitely (for more details, refer to the example readme file).

Note: *The principle of the STM32H750xx Value line applications is to execute the user application from an external memory (Quad-SPI Flash memory or SDRAM, by default Quad-SPI Flash memory). The Templates \ExtMem_Boot projects enables to boot from the STM32H750xx internal Flash memory, configure external memories, and then jump to the user application located in an external memory of the STM32H750B-DK board.*

Note: *The principle of the STM32H7B0xx Value line applications is to execute the user application from an external memory (Octo-SPI Flash memory or SDRAM, by default Octo-SPI Flash memory). The STM32H7B3I-DK \Applications\ExtMem_CodeExecution\ExtMem_Boot project enables to boot from the STM32H7B0xx internal Flash memory, configure external memories, and then jump to the user application located in an external memory of the STM32H7B3I-DK board.*

The following section provides a quick overview on how to open, build and run an example with the supported toolchains.

- EWARM
  1. Under the example folder, open the \\EWARM subfolder.
  2. Open the Project.eww workspace. The workspace name may change from one example to another.
  3. Rebuild all files: Project->Rebuild all.
  4. Load project image: Project->Debug.
  5. Run program: Debug->Go(F5).
- MDK-ARM
  1. Under the example folder, open the \\MDK-ARM subfolder.
  2. Open the Project.uvproj workspace. The workspace name may change from one example to another.
  3. Rebuild all files: Project->Rebuild all target files.
  4. Load project image: Debug->Start/Stop Debug Session.
  5. Run program: Debug->Run (F5).
- SW4STM32
  1. Open the SW4STM32 toolchain.
  2. Click File->Switch Workspace->Other and browse to the SW4STM32 workspace directory.
  3. Click File->Import, select General->'Existing Projects into Workspace' and then click “Next.
  4. Browse to the SW4STM32 workspace directory and select the project.
  5. Rebuild all project files: select the project in the “Project explorer” window then click on Project->build project menu.
- STM32CubeIDE
  1. Open the STM32CubeIDE toolchain.
  2. Click File->Switch Workspace->Other and browse to the STM32CubeIDE workspace directory.
  3. Click File->Import, select General->'Existing Projects into Workspace' and then click “Next.
  4. Browse to the STM32CubeIDE workspace directory and select the project.
  5. Rebuild all project files: select the project in the “Project explorer” window then click on Project->build project menu.

Note: *STM32CubeIDE projects are provided only for STM32H7B3I-EVAL, STM32H7B3I-DK and NUCLEO-H7A3ZI-Q boards*

4.2 Developing your own application

4.2.1 HAL application

This section describes the required steps needed to create your own application using STM32CubeH7.

1. **Create your project:** to create a new project you can either start from the Template project provided for each board under *\Projects\<STM32xx_xxx>\Templates* or from any available project under *\Projects\<STM32xx_xxx>\Examples* or *\Projects\<STM32xx_xxx>\Applications* (<STM32xx_xxx> refers to the board name, ex. STM32H743I_EVAL).

   The Template project provides an empty main loop function, it is a good starting point to get familiar with the project settings for STM32CubeH7. The template has the following characteristics:

   a. It contains sources of the HAL, CMSIS and BSP drivers which are the minimum required components to develop code for a given board
   b. It contains the include paths for all the firmware components
   c. It defines the STM32H7 device supported, allowing to have the right configuration for the CMSIS and HAL drivers
   d. It provides ready-to-use user files preconfigured as follows:
      - HAL is initialized
      - SysTick ISR implemented for HAL_Delay() purpose
      - System clock is configured with the maximum frequency of the device

Note: When copying an existing project to another location, make sure to update the include paths.

2. **Add the necessary middleware to your project (optional):** the available middleware stacks are: USB Host and Device Libraries, STemWin, LibJPEG, FreeRTOS™, FatFS, LwIP, and mbedTLS. To find out which source files you need to add to the project files list, refer to the documentation provided for each middleware, you may also have a look at the applications available under *\Projects\STM32xx_xxx\Applications\<MW_Stack>* (<MW_Stack> refers to the Middleware stack, for example USB_Device) to get a better idea of the source files to be added and the include paths.
3. **Configure the firmware components:** the HAL and middleware components offer a set of build time configuration options using macros declared with "#define" in a header file. A template configuration file is provided within each component, it has to be copied to the project folder (usually the configuration file is named *xxx_conf_template.h*. The word "_template" needs to be removed when copying it to the project folder). The configuration file provides enough information to know the effect of each configuration option. More detailed information is available in the documentation provided for each component.
4. **Start the HAL Library:** after jumping to the main program, the application code needs to call HAL_Init() API to initialize the HAL Library, which does the following:
   a. Configure the SysTick to generate an interrupt every 1ms. The SysTick is clocked by the HSI (default configuration after reset).
   b. Sets NVIC Group Priority to 4.
   c. Calls the HAL_MspInit() callback function defined in user file *stm32h7xx_hal_msp.c* to do the global low level hardware initialization.
5. **Configure the system clock:** the system clock configuration is done by calling the following APIs:
   a. HAL_RCC_OscConfig(): configures the internal and/or external oscillators, PLL source and factors. The user may select to configure one oscillator or all oscillators. The PLL configuration can be skipped if there is no need to run the system at high frequency.
   b. HAL_RCC_ClockConfig(): configures the system clock source, Flash latency and AHB and APB prescalers.
6. **Peripheral initialization**
   a. Start by writing the peripheral HAL_PPP_MspInit function. For this function, proceed as follows:
      - Enable the peripheral clock.
      - Configure the peripheral GPIOs.
      - Configure DMA channel and enable DMA interrupt (if needed).
      - Enable peripheral interrupt (if needed).
   b. Edit the *stm32h7xx_it.c* to call the required interrupt handlers (peripheral and DMA), if needed.
   c. Write process complete callback functions if you plan to use peripheral interrupt or DMA.
   d. In your *main.c* file, initialize the peripheral handle structure, then call the function HAL_PPP_Init() to initialize your peripheral.

7. **Develop your application process:** at this stage, your system is ready and you can start developing your application code.
   a. The HAL provides intuitive and ready-to-use APIs for configuring the peripheral, and supports polling, interrupt and DMA programming models, to accommodate any application requirements. For more details on how to use each peripheral, refer to the rich examples set provided.
   b. If your application has some real-time constraints, you can find a large set of examples showing how to use FreeRTOS and integrate it with all middleware stacks provided within STM32CubeH7, it can be a good starting point for your development.

Note: In the default HAL implementation, the SysTick timer is the timebase source. It is used to generate interrupts at regular time intervals. If HAL_Delay() is called from peripheral ISR process, the SysTick interrupt must have higher priority (numerically lower) than the peripheral interrupt. Otherwise, the caller ISR process is blocked. Functions affecting timebase configurations are declared as __Weak to make override possible in case of other implementations in user file (using a general purpose timer for example or other time source). For more details please refer to HAL_TimeBase example.

4.2.2 LL application

This section describes the steps needed to create your own LL application using STM32CubeH7.

1. Create your project

   To create a new project you either start from the Templates_LL project provided for each board under *\Projects\\Templates_LL* or from any available project under *\Projects\\Examples_LL* (refer to the board name, such as NUCLEO-H743ZI). The Template project provides an empty main loop function, however it is a good starting point to get familiar with project settings for STM32CubeH7.

   The Template main characteristics are the following:
   – It contains the source codes of the LL and CMSIS drivers which are the minimal components needed to develop code on a given board.
   – It contains the include paths for all the required firmware components.
   – It selects the supported STM32H7 device and allows to configure the CMSIS and LL drivers accordingly.
   – It provides ready-to-use user files, that are pre-configured as follows:
     main.c: system clock configuration for maximum frequency.
2. Port an existing project to another board

   To port an existing project to another target board, start from the Templates_LL project provided for each board and available under *\Projects\\Templates_LL*:
   a. Select a LL example

      To find the board on which LL examples are deployed, refer to the list of LL examples *STM32CubeProjectsList.html*, to Table 3: Number of examples for each board or to application note “*STM32Cube firmware examples for STM32H7 Series*” (AN5033)
   b. Port the LL example
      i. Copy/paste the *Templates_LL* folder (to keep the initial source, or directly update existing Templates_LL project).
      ii. Then porting consists principally in replacing *Templates_LL* files by the *Examples_LL* targeted project.

      Keep all board specific parts. For reasons of clarity, board specific parts have been flagged with specific tags:

```
/* ======== BOARD SPECIFIC CONFIGURATION CODE BEGIN ============== */
...
/* ============== BOARD SPECIFIC CONFIGURATION CODE END ========== */
```

      Thus the main porting steps are the following:
      1. Replace the *stm32h7xx_it.h* file
      2. Replace the *stm32h7xx_it.c* file
      3. Replace the *main.h* file and update it.
      4. Replace the *main.c* file and update it

   Thanks to these adaptations, the example should be functional on the targeted board.

4.3 Using STM32CubeMX to generate the initialization C code

An alternative to steps 1 to 6 described in Section 4.2 Developing your own application consists in using the STM32CubeMX tool to easily generate code for the initialization of the system, the peripherals and middleware (steps 1 to 6 above) through a step-by-step process:

1. Select the STMicroelectronics STM32 microcontroller that matches the required set of peripherals.
2. Configure each required embedded software thanks to a pinout-conflict solver, a clock-tree setting helper, a power consumption calculator, and an utility performing MCU peripheral configuration (GPIO, USART...) and middleware stacks (USB, TCP/IP...).
3. Generate the initialization C code based on the configuration selected. This code is ready to be used within several development environments. The user code is kept at the next code generation.

For more information, please refer to "*STM32CubeMX for STM32 configuration and initialization C code generation*" user manual (UM1718).

4.4 Getting STM32CubeH7 release updates

The STM32CubeH7 MCU Package comes with an updater utility: STM32CubeUpdater, also available as a menu within STM32CubeMX code generation tool.

The updater solution detects new firmware releases and patches available on *www.st.com* and proposes to download them to the user's computer.

4.4.1 Installing and running the STM32CubeUpdater program

1. Double-click SetupSTM32CubeUpdater.exe file to launch the installation.
2. Accept the license terms and follow the different installation steps.
3. Upon successful installation, STM32CubeUpdater becomes available as an STMicroelectronics program under Program Files and is automatically launched.

The STM32CubeUpdater icon appears in the system tray:

1. Right-click the updater icon and select Updater Settings to configure the Updater connection and to perform manual or automatic checks. For more details on Updater configuration, refer to section 3 of the STM32CubeMX User manual (UM1718).

5 FAQs

What is the license scheme for the STM32CubeH7 MCU Package?

The HAL is distributed under a non-restrictive BSD (Berkeley Software Distribution) license.

The middleware stacks made by ST (USB Host and Device Libraries, STemWin) come with a licensing model allowing easy reuse, provided it runs on an ST device.

The middleware based on well-known open-source solutions (FreeRTOS™, FatFs, LwIP and mbedTLS) have user-friendly license terms. For more details, refer to the license agreement of each middleware.

What boards are supported by the STM32CubeH7 MCU Package?

The STM32CubeH7 MCU Package provides BSP drivers and ready-to-use examples for the following STM32H7 boards: NUCLEO-H743ZI, NUCLEO-H745ZI-Q, STM32H743I-EVAL, STM32H745I-DISCO, STM32H747I-DISCO, STM32H747I-EVAL, STM32H750B-DK, STM32H7B3I-EVAL, STM32H7B3I-DK and NUCLEO-H7A3ZI-Q.

Does the HAL take benefit from interrupts or DMA? How can this be controlled?

Yes. The HAL supports three API programming models: polling, interrupt and DMA (with or without interrupt generation).

Are any examples provided with the ready-to-use toolset projects?

Yes. STM32CubeH7 provides a rich set of examples and applications (around 192 for STM32H743I-EVAL, 97 for STM32H747I-EVAL ...). They come with the preconfigured project of several toolsets: IAR, Keil and GCC.

How are the product/peripheral specific features managed?

The HAL offers extended APIs, i.e. specific functions as add-ons to the common API to support features available on some products/lines only.

How can STM32CubeMX generate code based on embedded software?

STM32CubeMX has a built-in knowledge of STM32 microcontrollers, including their peripherals and software. This enables the tool to provide a graphical representation to the user and generate *.h/*.c files based on user configuration.

How to get regular updates on the latest STM32CubeH7 firmware releases?

The STM32CubeH7 MCU Package comes with an updater utility, STM32CubeUpdater, that can be configured for automatic or on-demand checks for new MCU Package updates (new releases or/and patches).

STM32CubeUpdater is integrated as well within the STM32CubeMX tool. When using this tool for STM32H7 configuration and initialization C code generation, the user can benefit from STM32CubeMX self-updates as well as STM32CubeH7 MCU Package updates.

For more details, refer to Section 4.4 Getting STM32CubeH7 release updates.

Does the HAL/LL drivers support all STM32H7 lines single core, dual core and value line ?

Yes the HAL/LL drivers support all the STM32H7 lines:

- “DUAL_CORE” define is used to delimit code (defines , functions, macros..) available only in dual core line. This define is automatically available when user defines the right macro in *stm32h7xx.h* (see Table 1. Macros for STM32H7 Series).
- “CORE_CM4” or “CORE_CM7” defines are respectively used to delimit configuration/code specific to Cortex®-M4/Cortex®-M7 core. It shall be added by user into compiler preprocessor symbols for each target configuration (see Section 2.3 section "2.3 Level 2").
- Value line devices (STM32H750xx) are treated like single core devices.

What are the considerations to run a dual core example:

Make sure to compile and build both Cortex®-M7 and cortex®-M4 targets, then load the corresponding images into the the STM32H7 dual core device. Please refer to the *readme.txt* for each example.

What are the considerations to run value line example (STM32H750xx, STM32H750B-DK board):

Make sure to compile, build and load the ExtMem_Boot into the internal Flash memory. Make sure that the target external memory example (RAM/ROM) configuration is properly set in the ExtMem_Boot project *memory.h* file (DATA_AREA/CODE_AREA). Make sure to compile, build and load the application into the external memory. For more details please refer to the *readme.txt* of the STM32H750B-DK board ExtMem_Boot and Template_Project templates.

Revision history

Table 4. **Document revision history**

Date	Revision	Changes
26-Apr-2017	1	Initial release.
29-Aug-2017	2	Updated number of applications for both boards in Section 3.2 and Section 5 FAQs . Updated Figure 8 STM32CubeH7 example overview.
21-Jun-2018	3	Updated Introduction to replace STMCube™ by STM32Cube™ and update STM32Cube logo. Updated Section 1 STM32Cube main features. Firmware package replaced by MCU Package in the whole document. Changed STM32H743I_EVAL into STM32H743I-EVAL in the whole document when referring to the board name. Added STM32H750xx part numbers in Table 1 Macros for STM32H7 Series and Table 2 Evaluation, Discovery and Nucleo boards for STM32H7 Series. Updated number of STM32H743I-EVAL examples in Section 3.2 and Section 5 FAQs.
03-Jul-2018	4	Updated Figure 2 STM32CubeH7 firmware components.
03-Apr-2019	5	Major update consisting in the introduction of the low-layer (LL) API and affecting all sections of the document.
12-Nov-2019	6	Updated Section Introduction. Replaced QSPI Flash memory by Quad-SPI Flash memory. Added support for STM32H7A3/B3 devices through STM32H7B3I-EVAL, STM32H7B3I-DK and NUCLEO-H7A3ZI-Q boards.

Contents

List of tables

List of figures