UM1891
User manual

Getting started with STM32CubeF7 firmware package for STM32F7 Series

Introduction

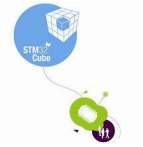
The STMCube™ initiative was originated by STMicroelectronics to ease developers life by reducing development efforts, time and cost. STM32Cube covers the STM32 portfolio.

The STM32Cube Version 1.x includes:

- The STM32CubeMX, a graphical software configuration tool that allows to generate C initialization code using graphical wizards
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF7 for STM32F7 Series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring a maximized portability across the STM32 portfolio
  - The Low Layer APIs (LL) offering a fast lightweight expert-oriented layer which is closer to the hardware than the HAL. The LL APIs are available only for a set of peripherals.
  - A consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
  - All embedded software utilities coming with a full set of examples.

This user manual describes how to get started with the STM32CubeF7 firmware package. *Section 1* describes the main features of the STM32CubeF7 firmware, part of the STMCube™ initiative.

Section 2 and *Section 3* provide an overview of the STM32CubeF7 architecture and firmware package structure.



Contents

1 STM32CubeF7 main features 5

2 STM32CubeF7 architecture overview 7

3 STM32CubeF7 firmware package overview 10

3.1 Supported STM32F7 Series devices and hardware 10

3.2 Firmware package overview 12

4 Getting started with STM32CubeF7 15

4.1 Running your first example 15

4.2 Developing your own application 16

4.2.1 HAL application 16

4.2.2 LL application 18

4.3 Using STM32CubeMX to generate the initialization C code 19

4.4 Getting STM32CubeF7 release updates 19

4.4.1 Installing and running the STM32CubeUpdater program 19

5 FAQ 20

6 Revision history 22

List of tables

Table 1. Macros for STM32F7 Series 10
Table 2. Evaluation and discovery boards for STM32F7 Series 11
Table 3. Number of examples available for each board 13
Table 4. Document revision history 22

List of figures

Figure 1. STM32CubeF7 firmware components 6
Figure 2. STM32CubeF7 firmware architecture 7
Figure 3. STM32CubeF7 firmware package structure 12
Figure 4. STM32CubeF7 example overview 14

1 STM32CubeF7 main features

STM32CubeF7 gathers together, in a single package, all the generic embedded software components required to develop an application on STM32F7 microcontrollers. In line with the STMCube™ initiative, this set of components is highly portable, not only within the STM32F7 Series but also to other STM32 Series.

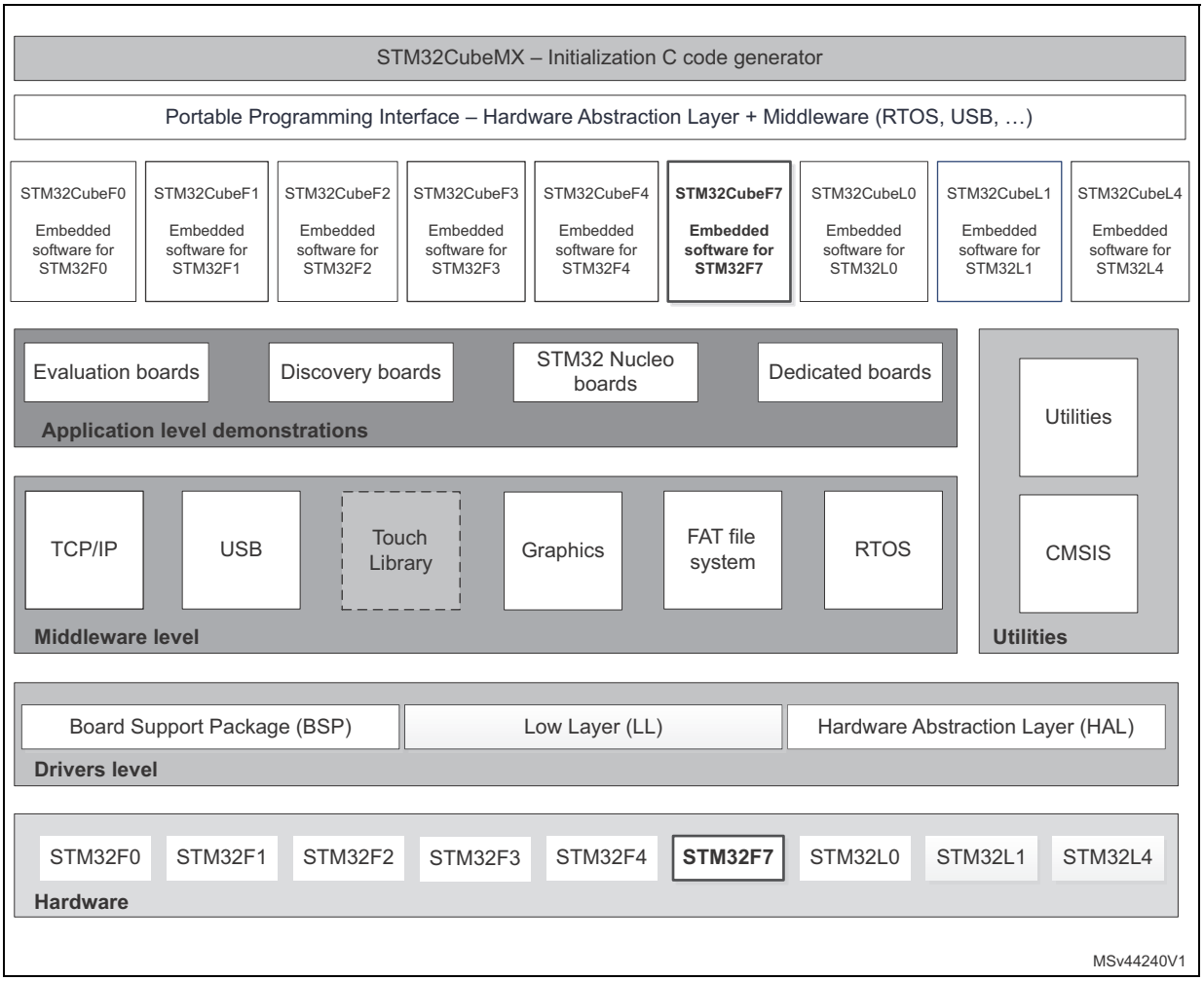
STM32CubeF7 is fully compatible with STM32CubeMX code generator that allows the user to generate initialization code. The package includes the Low Layer (LL) and the hardware abstraction layer (HAL) APIs that cover the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL and LL APIs are available in an open-source BSD license for user convenience.

The STM32CubeF7 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.
- CMSIS-RTOS implementation with FreeRTOS open source solution
- 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 PolarSSL

A demonstration implementing all these middleware components is also provided in the STM32CubeF7 package.

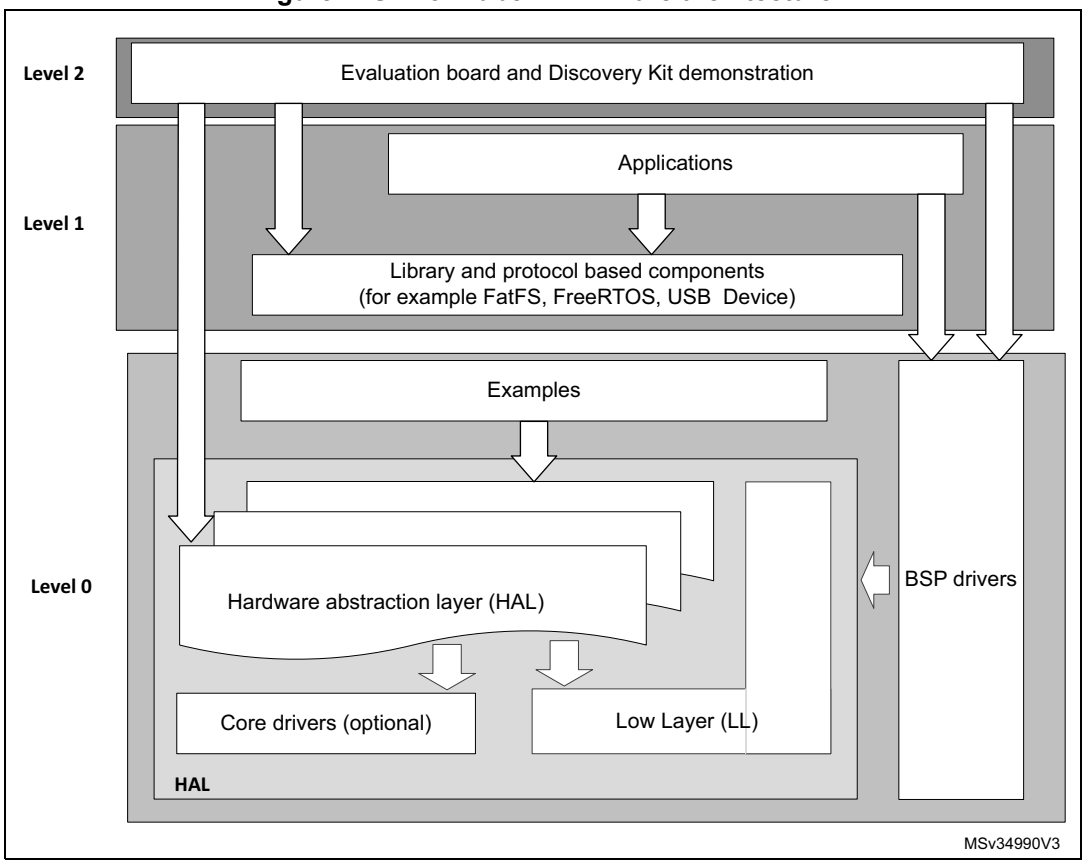
Figure 1. STM32CubeF7 firmware components



2 STM32CubeF7 architecture overview

The STM32CubeF7 firmware solution is built around three independent levels that can easily interact with each other as described in *Figure 2*.

Figure 2. STM32CubeF7 firmware architecture



Level 0: This level is divided into three sub-layers:

- **Board Support Package (BSP):** this layer offers a set of APIs relative to the hardware components in the hardware boards (Audio codec, I/O expander, Touchscreen, SRAM driver, LCD drivers. etc…). 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 permits to link 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()

  It is based on a modular architecture allowing an easy porting on any hardware by just implementing the low level routines.
- **Hardware Abstraction Layer (HAL):** this 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.

- **Basic peripheral usage examples:** this layer contains the examples of the basic operation of the STM32F7 peripherals using either the HAL or/and the Low Layer drivers APIs as well as the BSP resources.
- **Low Layer (LL)**:
  - The low layer APs provide low-level APIs at register level, with a better optimization but less portability. They require a deep knowledge of MCU and peripheral specifications. The LL drivers are designed to offer a fast lightweight expert-oriented layer which is closer to the hardware than the HAL. Contrary to the HAL, the LL APIs are not provided for peripherals where the optimized access is not a key feature, or those requiring a heavy software configuration and/or a complex upper-level stack (such as FMC, USB or SDMMC).

  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
  - A function for peripheral de-initialization (peripheral registers restored to their default values)
  - A set of inline functions for a direct and atomic register access
  - A full independence from the HAL and the capability to be used in standalone mode (without HAL drivers)
  - A full coverage of the supported peripheral features.

Level 1: This level is divided into two sub-layers:

- **Middleware components:** a set of Libraries covering USB Host and Device Libraries, STemWin, LibJPEG, FreeRTOS, FatFS, LwIP, and PolarSSL. 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 multi packet 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.

  FreeRTOS
  - Open source standard
  - CMSIS compatibility layer
  - Tickless operation during low-power mode
  - Integration with all STM32Cube middleware modules

  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

- **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.

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.

3 STM32CubeF7 firmware package overview

3.1 Supported STM32F7 Series 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 STM32CubeF7 offers a full support for all the STM32F7 Series devices. The user only needs to define the right macro in stm32f7xx.h.

Table 1 lists which macro to define depending on the used STM32F7 Series device. This macro can also be defined in the compiler preprocessor.

Table 1. Macros for STM32F7 Series

Macro defined in stm32f7xx.h	STM32F7 Series devices
STM32F756xx	STM32F756VG, STM32F756ZG, STM32F756IG, STM32F756BG, STM32F756NG
STM32F746xx	STM32F746VE, STM32F746VG, STM32F746ZE, STM32F746ZG, STM32F746IE, STM32F746IG, STM32F746BE, STM32F746BG, STM32F746NE, STM32F746NG
STM32F745xx	STM32F745VE, STM32F745VG, STM32F745ZG, STM32F745ZE, STM32F745IE, STM32F745IG
STM32F765xx	STM32F765BI, STM32F765BG, STM32F765NI, STM32F765NG, STM32F765II, STM32F765IG, STM32F765ZI,STM32F765ZG, STM32F765VI, STM32F765VG
STM32F767xx	STM32F767BG, STM32F767BI, STM32F767IG, STM32F767II, STM32F767NG, STM32F767NI, STM32F767VG, STM32F767VI, STM32F767ZG, STM32F767ZI, STM32F768AI
STM32F769xx	STM32F769AG, STM32F769AI, STM32F769BG, STM32F769BI, STM32F769IG, STM32F769II, STM32F769NG, STM32F769NI
STM32F777xx	STM32F777BI, STM32F777II, STM32F777NI, STM32F777VI, STM32F777ZI, STM32F778AI
STM32F779xx	STM32F779AI, STM32F779BI, STM32F779II, STM32F779NI
STM32F722xx	STM32F722IE, STM32F722ZE, STM32F722VE, STM32F722RE, STM32F722IC, STM32F722ZC, STM32F722VC, STM32F722RC
STM32F723xx	STM32F723IE, STM32F723ZE, STM32F723VE, STM32F723IC, STM32F723ZC, STM32F723VC
STM32F732xx	STM32F732IE, STM32F732ZE, STM32F732VE, STM32F732RE
STM32F733xx	STM32F733IE, STM32F733ZE, STM32F733VE

STM32CubeF7 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*:

Table 2. Evaluation and discovery boards for STM32F7 Series

Board	STM32F7 Series devices supported
STM327x6G_EVAL(1)	STM32F746xx and STM32F756xx
STM32F746G-Discovery	STM32F746NG
STM32F746ZG-Nucleo	STM32F746ZG
STM32F7x9I_EVAL(2)	STM32F779xx and STM32F769xx
STM32F769I-Discovery	STM32F769NI
STM32F767ZI-Nucleo	STM32F767ZI
STM32F723E-Discovery	STM32F723IE
STM32F722ZE-Nucleo	STM32F722ZE

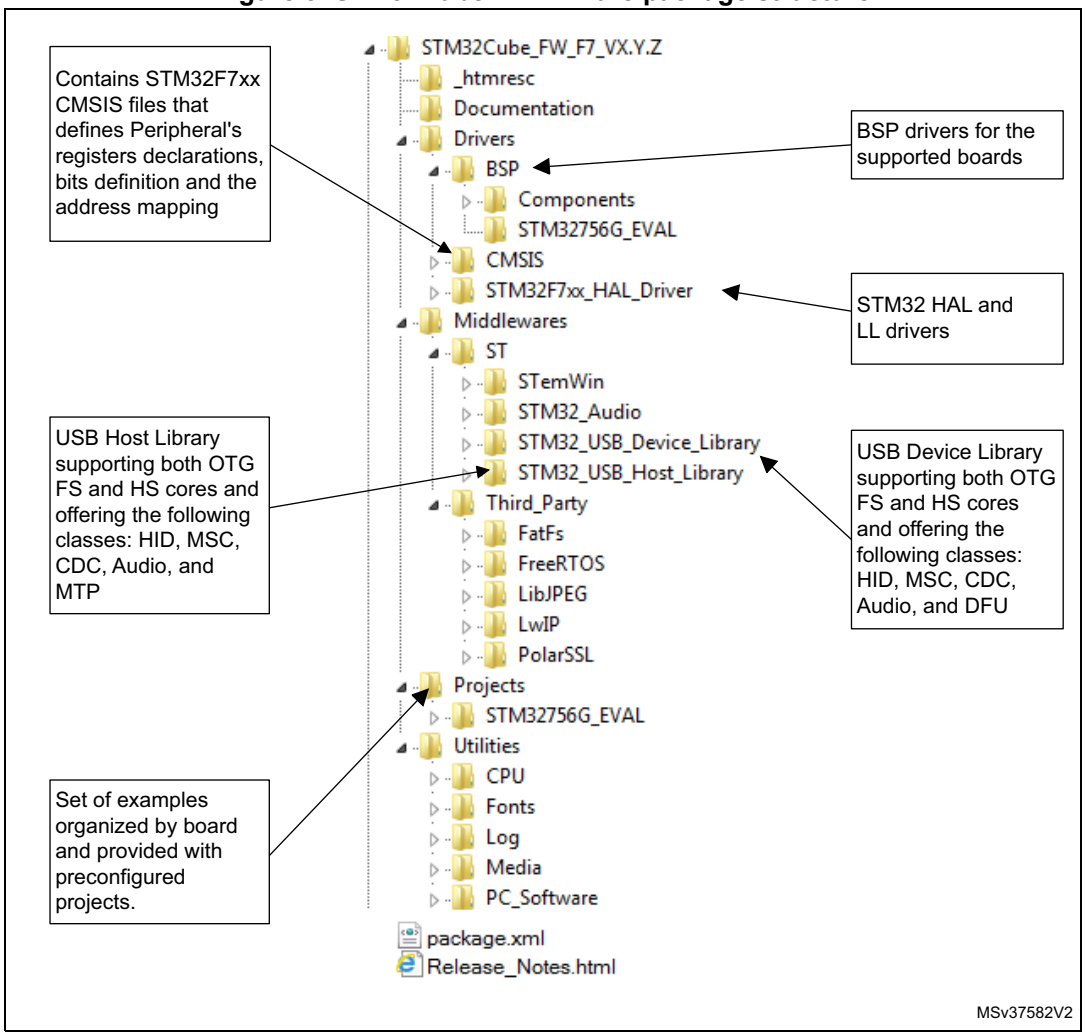
1. STM327x6G_EVAL refers to STM32746G_EVAL and STM32756G_EVAL evaluation boards.
2. STM32F7x9I_EVAL refers to STM32F769I_EVAL and STM32F779I_EVAL evaluation boards.

The STM32CubeF7 firmware 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 Firmware package overview

The STM32CubeF7 firmware solution is provided in a single zip package with the structure shown in *Figure 3*.

Figure 3. STM32CubeF7 firmware package structure



For each board, a set of examples are provided with preconfigured projects for EWARM, MDK-ARM and SW4STM32 toolchains.

Figure 4 shows the project structure for the STM327x6G_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, Examples_LL, and Examples_MIX. They use respectively HAL drivers, LL drivers and a mix of HAL and LL 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

A template project is provided to allow to quickly build any firmware application on a given board.

All examples have the same structure,

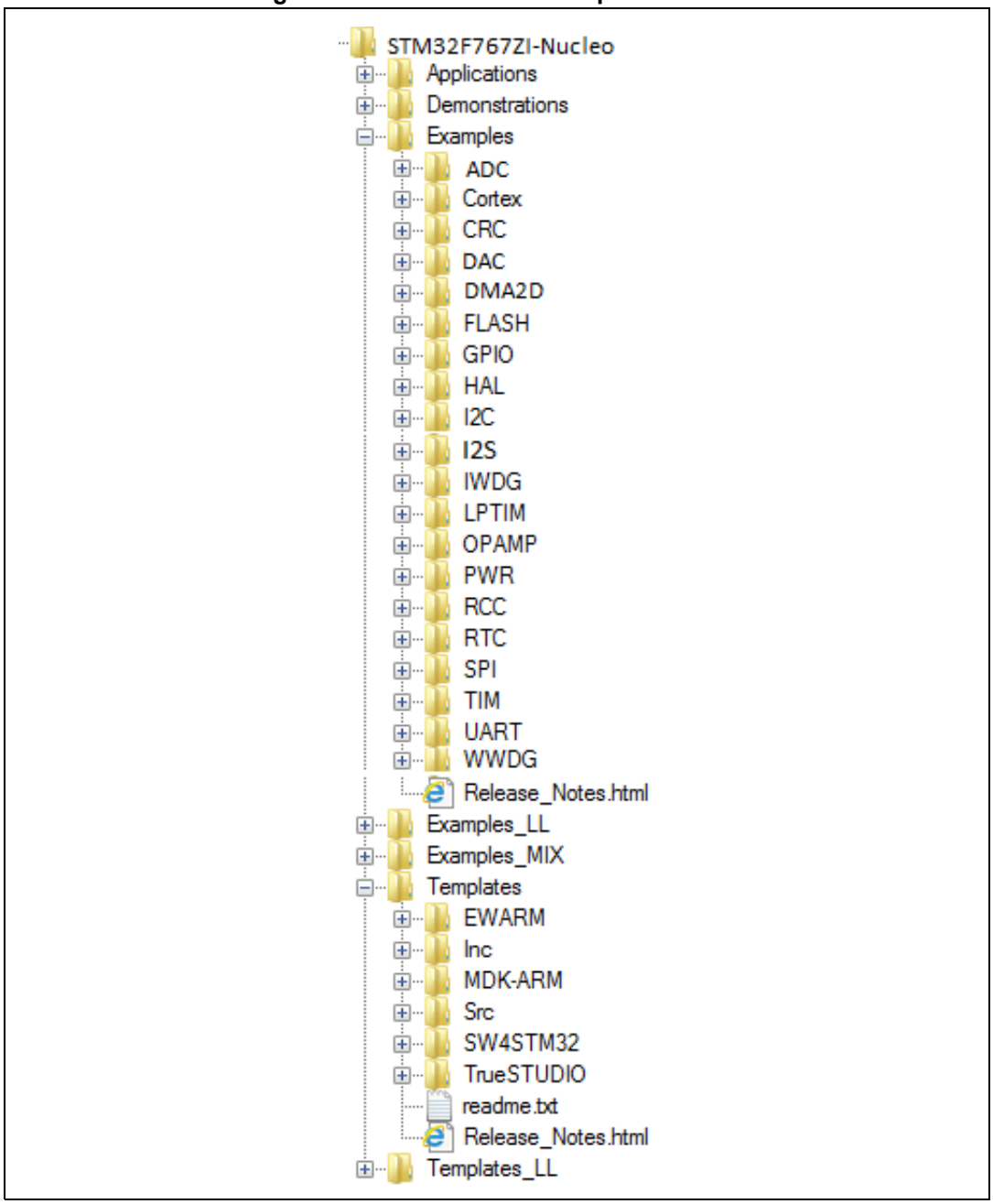
- \Inc folder that contains all header files
- \Src folder for the sources code
- \EWARM, \MDK-ARM and \SW4STM32 folders contain the preconfigured project for each toolchain.
- readme.txt describing the example behavior and the environment required to make it work

Table 3 provides the number of examples, applications and demonstrations available for each board.

Table 3. Number of examples available for each board

Board	Templates _LL	Examples	Examples _LL	Examples _MIX	Applications	Demonstration
STM327x6G_EVAL	0	93	0	0	63	1
STM32F746G-Discovery	0	32	0	0	26	1
STM32F746ZG-Nucleo	0	28	0	0	8	1
STM32F7x9I_EVAL	0	123	0	0	48	1
STM32F769I-Discovery	0	30	0	0	16	1
STM32F767ZI-Nucleo	1	41	73	15	8	1
STM32F723E-Discovery	0	41	0	0	24	1
STM32F722ZE-Nucleo	0	33	0	0	17	1

Figure 4. STM32CubeF7 example overview



4 Getting started with STM32CubeF7

4.1 Running your first example

This section explains how simple it is to run a first example with STM32CubeF7. It uses as an illustration the generation of a simple LED toggling example running on the STM327x6G_EVAL board:

1. After downloading the STM32CubeF7 firmware package, unzip it into a directory of your choice, make sure not to modify the package structure shown in *Figure 3*.
2. Browse to \Projects\STM327x6G_EVAL\Examples.
3. Open \GPIO, then the \GPIO_EXTI folder.
4. Open the project with your preferred toolchain.
5. Rebuild all files and load your image into target memory.
6. Run the example: each time you press the Tamper push-button, the LED1 will toggle (for more details, refer to the example readme file).

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

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

a. The workspace name may change from one example to another.

4.2 Developing your own application

4.2.1 HAL application

This section describes the required steps needed to create your own application using STM32CubeF7.

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. STM327x6G_EVAL).

   The Template project provides an empty main loop function, it is a good starting point to get familiar with the project settings for STM32CubeF7. 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 STM32F7 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 PolarSSL. 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 Flash prefetch, and instruction cache through ART accelerator.

   b) Configure the SysTick to generate an interrupt every 1ms. The SysTick is clocked by the HSI (default configuration after reset)

   c) Sets NVIC Group Priority to 4

   d) Calls the HAL_MspInit() callback function defined in user file stm32f7xx_hal_msp.c to do the global low level hardware initialization

5. **Configure the system clock:** the system clock configuration is done by calling these two 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 stm32f7xx_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 in STM32CubeF7, 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 STM32CubeF7.

1. **Create your project**

   To create a new project, it is possible to start either from the *Templates_LL* project provided for each board under \Projects\<STM32xxx_yyy>\Templates_LL or from any available project under \Projects\<STM32xxy_yyy>\Examples_LL (<STM32xxx_yyy> refers to the board name, such as STM32F767ZI-Nucleo).

   The *Template* project provides an empty main loop function, however it is a good starting point to get familiar with project settings for STM32CubeF7.

   The template main characteristics are listed below:

   a) It contains the source code of LL and CMSIS drivers, that are the minimal components to develop a code on a given board.

   b) It contains the include paths for all the required firmware components.

   c) It selects the supported STM32F7 device and allows configuring the CMSIS and LL drivers accordingly.

   d) It provides ready-to-use user files, that are pre-configured as follows:

   - main.h: LED & USER_BUTTON definition abstraction layer

   - main.c: System clock configured with the 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\<STM32xxx_yyy>\Templates_LL:

   a) Select an LL example

   To find the board on which LL examples are deployed, refer to the list of LL examples in STM32CubeProjectsList.html, to *Table 3: Number of examples available for each board*, or to *STM32Cube firmware examples for STM32F7 Series* application note (AN4731).

   b) Port the LL example

   – Copy/paste the Templates_LL folder to keep the initial source or directly update the existing *Templates_LL* project.

   – Then LL example porting consists mainly in replacing the *Templates_LL* files by the *Examples_LL* targeted.

   – Keep all board specific parts. For reasons of clarity, the 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:

   – Replace stm32f7xx_it.h file

   – Replace stm32f7xx_it.c file

   – Replace main.h file and update it: keep the LED and user button definition of the LL template under "BOARD SPECIFIC CONFIGURATION" tags.

   – Replace main.c file, and update it:

   - Keep the clock configuration of the SystemClock_Config() LL template: function under "BOARD SPECIFIC CONFIGURATION" tags.

- Depending on LED definition, replace all LEDx_PIN by another LEDx (number) available in main.h file.

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* 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 STM32CubeF7 release updates

The STM32CubeF7 firmware 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

- Double-click SetupSTM32CubeUpdater.exe file to launch the installation.
- Accept the license terms and follow the different installation steps.

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:

- 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 FAQ

What is the license scheme for the STM32CubeF7 firmware?

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 PolarSSL) have user-friendly license terms. For more details, refer to the license agreement of each middleware.

What boards are supported by the STM32CubeF7 firmware package?

The STM32CubeF7 firmware package provides BSP drivers and ready-to-use examples for the following STM32F7 boards: STM327x6G_EVAL, STM32F746G-Discovery, STM32F746ZG-Nucleo, STM32F769I_EVAL, STM32F769I-Discovery, STM32F723E-Discovery, STM32F722ZE-Nucleo and STM32F767ZI-Nucleo.

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. STM32CubeF7 provides a rich set of examples and applications (around 150 for STM327x6G_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 STM32CubeF7 firmware releases?

The STM32CubeF7 firmware package comes with an updater utility, STM32CubeUpdater, that can be configured for automatic or on-demand checks for new firmware package updates (new releases or/and patches).

STM32CubeUpdater is integrated as well within the STM32CubeMX tool. When using this tool for STM32F7 configuration and initialization C code generation, the user can benefit from STM32CubeMX self-updates as well as STM32CubeF7 firmware package updates.

For more details, refer to *Section 4.4*.

Is there any link with standard peripheral libraries?

The STM32Cube HAL and LL drivers are the replacement of the standard peripheral library:

- The HAL drivers offer a higher abstraction level compared to the standard peripheral APIs. They focus on peripheral common features rather than hardware. Their higher abstraction level allows defining a set of user-friendly APIs that can be easily ported from one product to another.
- The LL drivers offer low-level APIs at register level. They are organized in a simpler and clearer way than direct register accesses. The LL drivers also include peripheral initialization APIs, which are more optimized compared to what is offered by the SPL, while being functionally similar. Compared to the HAL drivers, these LL initialization APIs allows an easier migration from the SPL to the STM32Cube LL drivers, since each SPL API has its equivalent LL API(s).

How can I include LL drivers in my environment? Is there any LL configuration file as for HAL?

There is no configuration file. The source code shall directly include the necessary stm32f7xx_ll_ppp.h file(s).

Can I use HAL and LL drivers together? If yes, what are the constraints?

It is possible to use both HAL and LL drivers. One can handle the IP initialization phase with the HAL and then manage the I/O operations with the LL drivers.

The major difference between HAL and LL is that the HAL drivers require to create and use handles for operation management while the LL drivers operates directly on peripheral registers. Mixing HAL and LL is illustrated in the Examples_MIX example.

When should I use HAL versus LL drivers?

The HAL drivers offer high-level and function-oriented APIs, with a high level of portability. the product/IPs complexity is hidden for end users.

The LL drivers offer low-level APIs at registers level, with a better optimization but less portability. They require a deep knowledge of the product/IPs specifications.

Is there any LL APIs which are not available with HAL?

Yes, there are.

A few Cortex® APIs have been added in stm32f7xx_ll_cortex.h e.g. for accessing the SCB or the SysTick registers.

How are LL initialization APIs enabled?

The definition of LL initialization APIs and associated resources (structure, literals and prototypes) is conditioned by the USE_FULL_LL_DRIVER compilation switch.

To be able to use the LL APIs, add this switch in the toolchain compiler preprocessor.

Why are SysTick interrupts not enabled on LL drivers?

When using the LL drivers in standalone mode, the user does not need to enable SysTick interrupts because they are not used in the LL APIs, while the HAL functions requires SysTick interrupts to manage timeouts.

6 Revision history

Table 4. Document revision history

Date	Revision	Changes
30-Apr-2015	1	Initial release.
12-Nov-2015	2	Updated *Table 2: Evaluation and discovery boards for STM32F7 Series* title and adding 2 new rows for STM32F746G-Discovery and STM32F746ZG-Nucleo boards. Updated *Table 3: Number of examples available for each board* application number at 61 instead of 58 and adding 2 new rows for STM32F746G-Discovery and STM32F746ZG-Nucleo boards. Updated *Figure 1: STM32CubeF7 firmware components*. Updated *Section 5: FAQ* 2nd question about the boards supported by STM32CubeF7 firmware package, adding STM32F746G-Discovery and STM32F746ZG-Nucleo boards.
21-Apr-2016	3	Updated *Table 1: Macros for STM32F7 Series* adding rows for STM32F76xxx and STM32F77xxx devices. Updated *Table 2: Evaluation and discovery boards for STM32F7 Series* adding new boards and note 2. Updated *Table 3: Number of examples available for each board*.
21-Dec-2016	4	Added low layer API (LL) feature: – Updated cover – Updated *Section 1: STM32CubeF7 main features*. – Updated *Figure 1: STM32CubeF7 firmware components*. – Updated *Section 2: STM32CubeF7 architecture overview*. – Updated *Figure 2: STM32CubeF7 firmware architecture*. Updated *Table 1: Macros for STM32F7 Series* adding macros for STM32F72xxx and STM32F73xxx devices. Added STM32F723E-Discovery and STM32F722ZE-Nucleo boards: – Updated *Table 2: Evaluation and discovery boards for STM32F7 Series*. – Updated *Table 3: Number of examples available for each board*. Updated *Section 4: Getting started with STM32CubeF7* adding *Section 4.2.2: LL application*. Updated *Section 5: FAQ*.