sizes, respectively. These values may need to be increased when the application uses middleware stacks.

- Firmware package selection when more than one version is available (this is the case when successive versions implement the same API and support the same MCUs). By default, the latest available version is used.

4.8.2 Code Generator tab

The Code Generator tab allows specifying the following code generation options (see *Figure 40*):

- STM32Cube Firmware Library Package option
- Generated files options
- HAL settings options
- Custom code template options

STM32Cube Firmware Library Package option

The following actions are possible:

- Copy all used libraries into the project folder

  STM32CubeMX will copy to the user project folder, the drivers libraries (HAL, CMSIS) and the middleware libraries relevant to the user configuration (e.g. FatFs, USB, ..).
- *Copy only the necessary library files*:

  STM32CubeMX will copy to the user project folder only the library files relevant to the user configuration (e.g., SDIO HAL driver from the HAL library,…).
- *Add the required library as referenced in the toolchain project configuration file*

  By default, the required library files are copied to the user project. Select this option for the configuration file to point to files in STM32CubeMX repository instead: the user project folder will not hold a copy of the library files but only a reference to the files in STM32CubeMX repository.

Generated files options

This area allows defining the following options:

- Generate peripheral initialization as a pair of .c/.h files or keep all peripheral initializations in the main.c file.
- Backup previously generated files in a backup directory

  The .bak extension is added to previously generated .c/.h files.

  Keep user code when regenerating the C code.

  This option applies only to user sections within STM32CubeMX generated files. It does not apply to the user files that might have been added manually or generated via ftl templates.
- Delete previously generated files when these files are no longer needed by the current configuration. For example, uart.c/.h file are deleted if the UART peripheral, that was enabled in previous code generation, is now disabled in current configuration.

HAL settings options

This area allows selection one HAL settings options among the following:

- Set all free pins as analog to optimize power consumption
- Enable/disable Use the *Full Assert* function: the Define statement in the stm32xx_hal_conf.h configuration file will be commented or uncommented, respectively.

Custom code template options

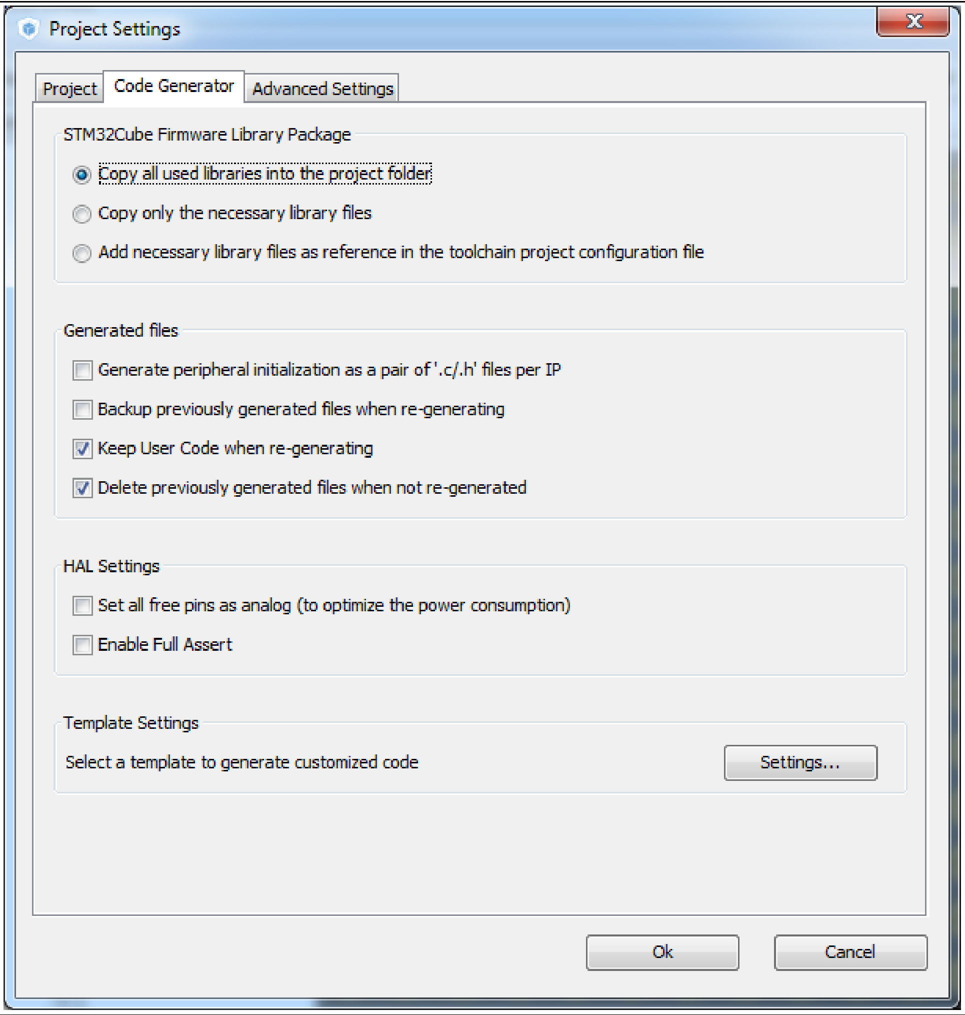
To generate custom code, click the Settings button under Template Settings, to open the Template Settings window (see *Figure 41*).

The user will then be prompted to choose a source directory to select the code templates from, and a destination directory where the corresponding code will be generated.

The default source directory points to the extra_template directory, within STM32CubeMX installation folder, which is meant for storing all user defined templates. The default destination folder is located in the user project folder.

STM32CubeMX will then use the selected templates to generate user custom code (see *Section 5.2: Custom code generation*). *Figure 42* shows the result of the template configuration shown on *Figure 41*: a sample.h file is generated according to sample_h.ftl template definition.

Figure 40. Project Settings Code Generator



Project Settings

Project | Code Generator | Advanced Settings

STM32Cube Firmware Library Package

- (●) Copy all used libraries into the project folder
- ( ) Copy only the necessary library files
- ( ) Add necessary library files as reference in the toolchain project configuration file

Generated files

- [ ] Generate peripheral initialization as a pair of '.c/.h' files per IP
- [ ] Backup previously generated files when re-generating
- [✓] Keep User Code when re-generating
- [✓] Delete previously generated files when not re-generated

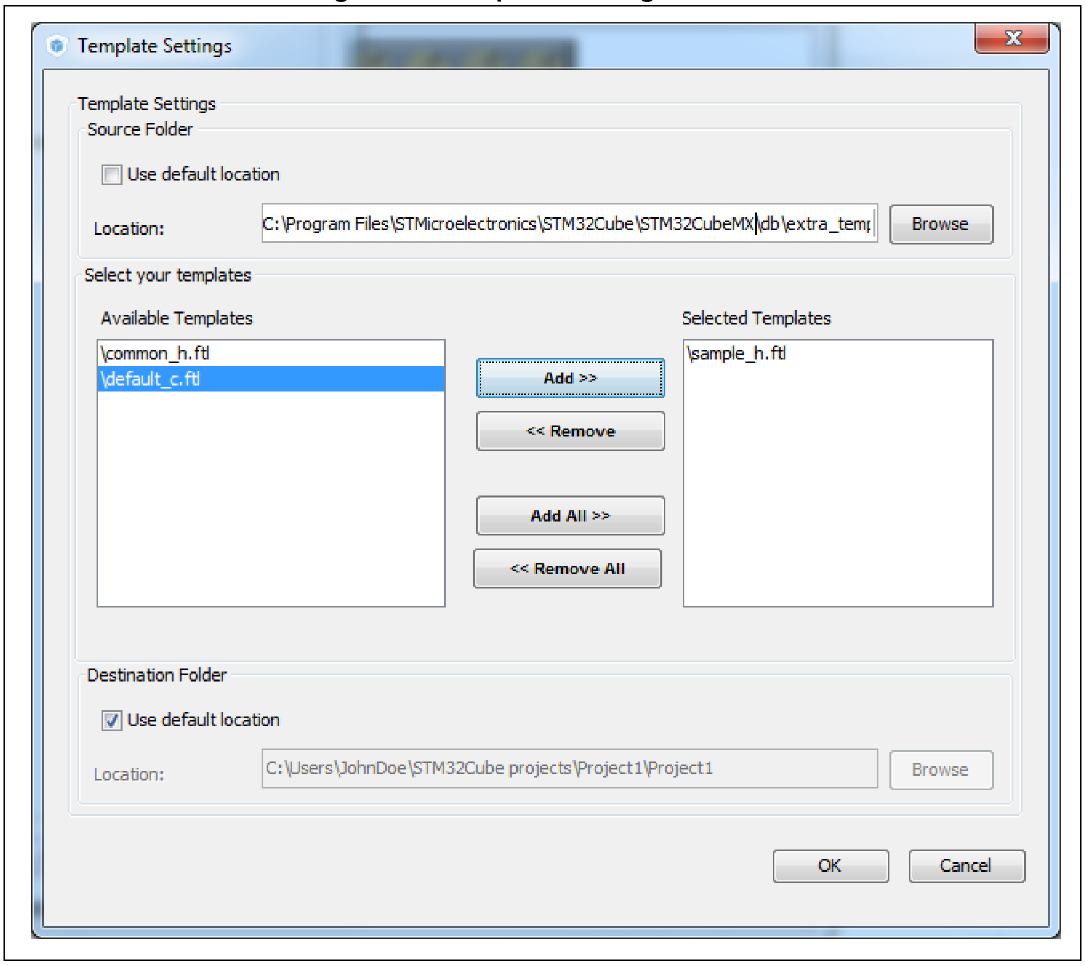
HAL Settings

- [ ] Set all free pins as analog (to optimize the power consumption)
- [ ] Enable Full Assert

Template Settings

Select a template to generate customized code

Figure 41. Template Settings window


Template Settings

Template Settings

Source Folder

☐ Use default location

Location: C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeMX\db\extra_temp

Select your templates

Available Templates: \common_h.ftl, \default_c.ftl

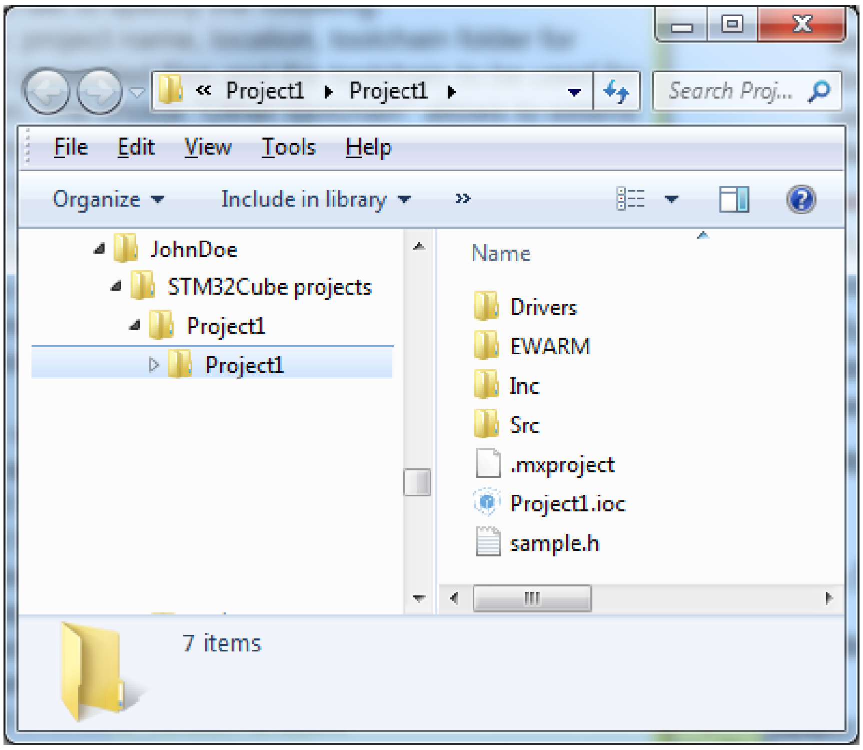
Selected Templates: \sample_h.ftl

Destination Folder

☑ Use default location

Location: C:\Users\JohnDoe\STM32Cube projects\Project1\Project1

Figure 42. Generated project template

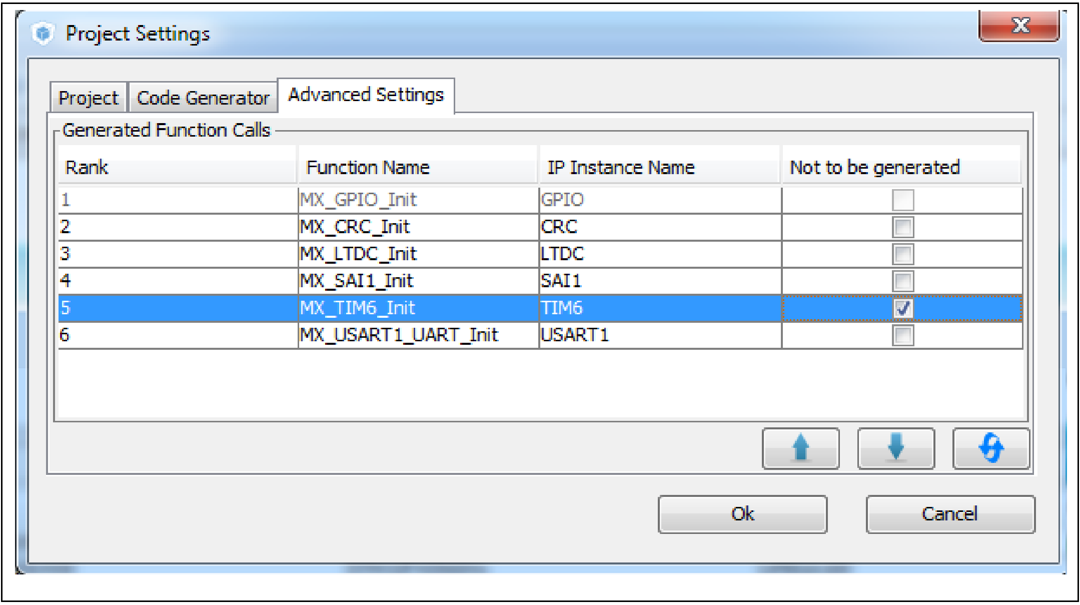


4.8.3 Advanced Settings tab

Figure 43 shows the case of several peripheral and/or middleware selections. By default the peripheral/middleware initialization functions are called in the order in which they have been enabled. The user can choose to re-order them by modifying the Rank number using the up and down arrow buttons. A reset button allows switching back to alphabetical order. If the Not to be generated checkbox is checked, STM32CubeMX does not generate the call to the peripheral initialization function. It is up to the user code to do it.

Note: Useful tooltips are also available by hovering the mouse over the different options.

Figure 43. Advanced Settings window



Project Settings

Project | Code Generator | Advanced Settings

Generated Function Calls

Rank	Function Name	IP Instance Name	Not to be generated
1	MX_GPIO_Init	GPIO	☐
2	MX_CRC_Init	CRC	☐
3	MX_LTDC_Init	LTDC	☐
4	MX_SAI1_Init	SAI1	☐
5	MX_TIM6_Init	TIM6	✓
6	MX_USART1_UART_Init	USART1	☐

4.9 Update Manager windows

Three windows can be accessed through the Help menu available from STM32CubeMX menu bar:

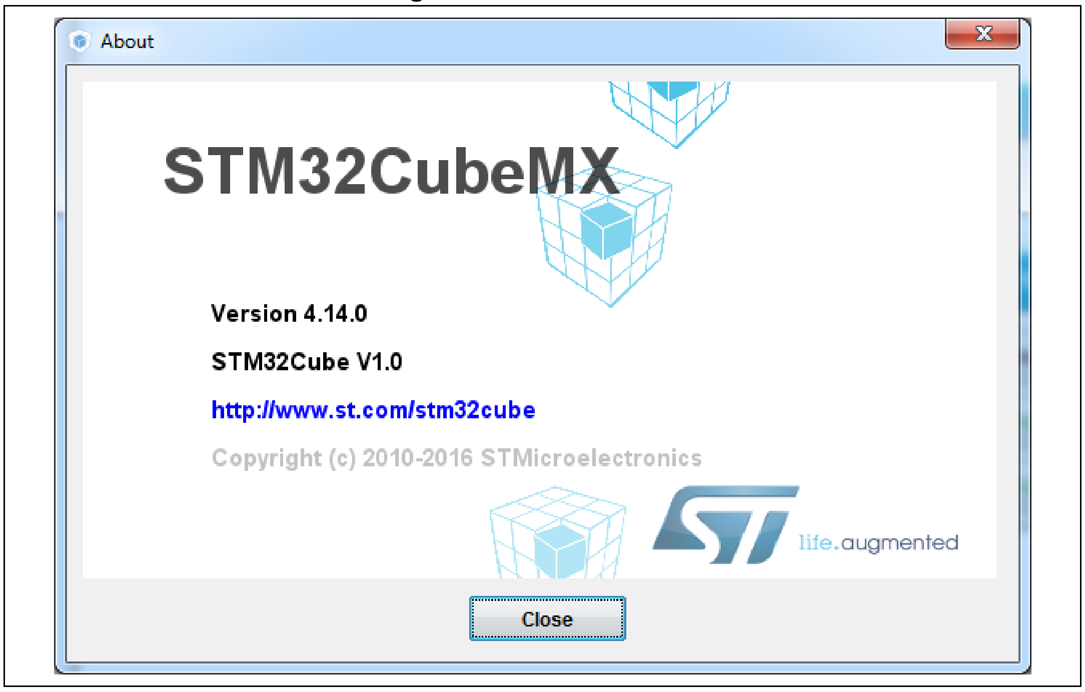
1. Select **Help > Check for updates** to open the **Check Update Manager** window and find out about the latest software versions available for download.
2. Select **Help > Install new libraries** to open the **New Libraries Manager** window and find out about the software packages available for download. It also allows removing previously installed software packages.
3. Select **Help > Updater settings** to open the **Updater settings** window and configure update mechanism settings (proxy settings, manual versus automatic updates, repository folder where STM32Cube software packages are stored).

4.10 About window

This window displays STM32CubeMX version information.

To open it, select **Help** > **About** from the STM32CubeMX menu bar.

Figure 44. About window



Version 4.14.0

STM32Cube V1.0

http://www.st.com/stm32cube

Copyright (c) 2010-2016 STMicroelectronics

4.11 Pinout view

The **Pinout** view helps the user configuring the MCU pins based on a selection of peripherals/middleware and of their operating modes.

Note: For some middleware (USB, FATS, LwIP), a peripheral mode must be enabled before activating the middleware mode. Tooltips guide the user through the configuration.

For FatFs, a user-defined mode has been introduced. This allows STM32CubeMX to generate FatFs code without a predefined peripheral mode. Then, it will be up to the user to connect the middleware with a user-defined peripheral by updating the generated user_diskio.c/.h driver files with the necessary code.

Since STM32 MCUs allow a same pin to be used by different peripherals and for several functions (alternate functions), the tool searches for the pinout configuration that best fits the set of peripherals selected by the user. STM32CubeMX highlights the conflicts that cannot be solved automatically.

The **Pinout** view left panel shows the **IP tree** and the right pane, a graphical representation of the pinout for the selected package (e.g. BGA, QFP...) where each pin is represented with its name (e.g. PC4) and its current alternate function assignment if any.

STM32CubeMX offers two ways to configure the microcontroller:

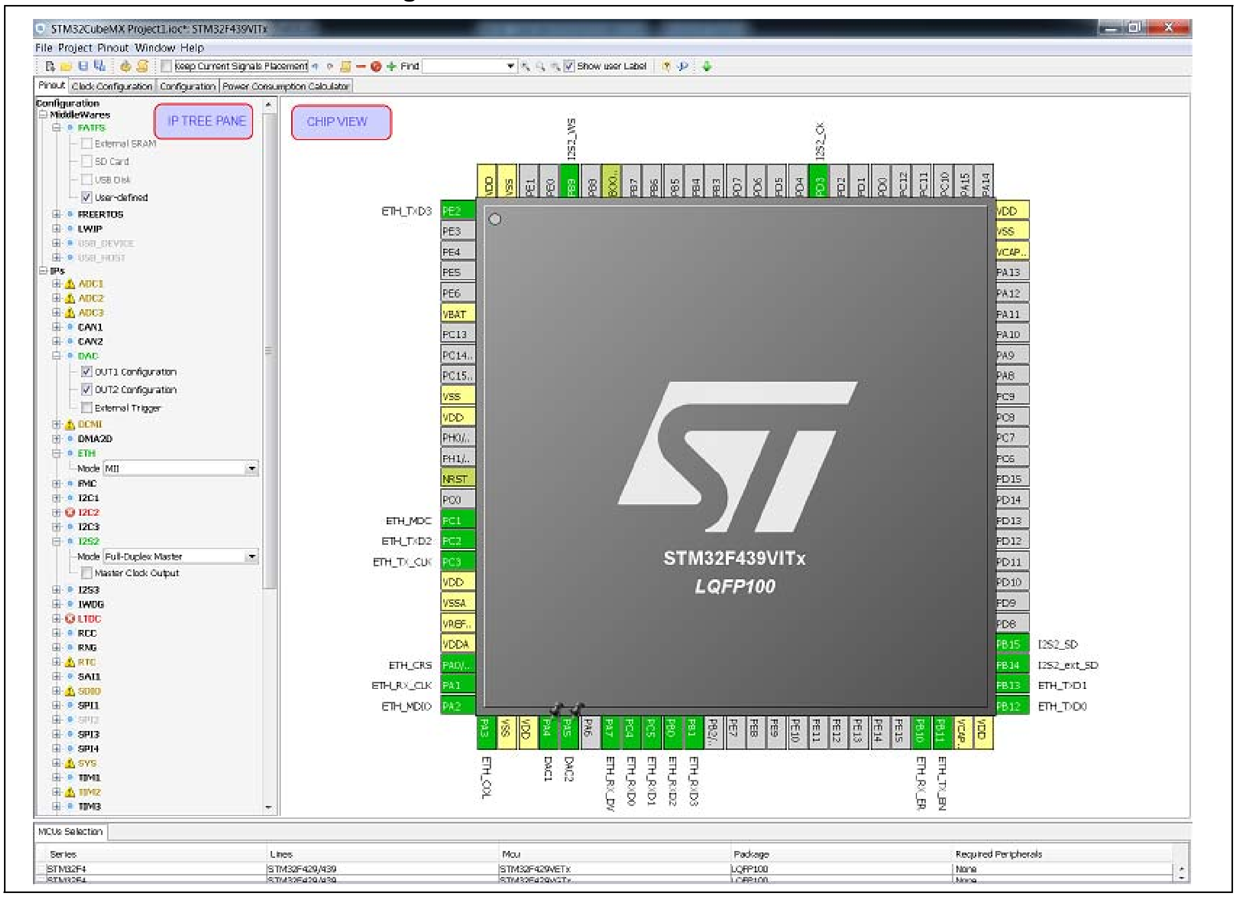
- From the **IP tree** by clicking the peripheral names and selecting the operating modes (see *Section 4.11.1: IP tree pane*).
- For advanced users, by clicking a pin on the **Chip** view to manually map it to a peripheral function (see *Section 4.11.2: Chip view*).

In addition, selecting **Pinout > Set unused GPIOs** allows configuring in one shot several unused pins in a given GPIO mode.

*Note: The **Pinout** view is automatically refreshed to display the resulting pinout configuration.*

*Pinout relevant menus and shortcuts are available when the **Pinout** view is active (see the menu dedicated sections for details on the **Pinout** menus).*

Figure 45. STM32CubeMX Pinout view



4.11.1 IP tree pane

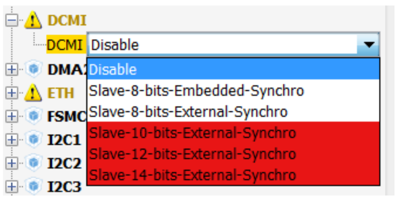
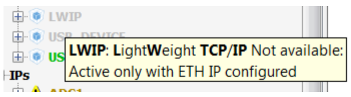
In this pane, the user can select the peripherals, services (DMA, RCC,...), middleware in the modes corresponding to the application.

Note: The peripheral tree panel is also accessible from the ***Configuration*** *view. However, only the peripherals and middleware modes without influence on the pinout can be configured through this menu.*

Icons and color schemes

Table 8 shows the icons and color scheme used in the **IP tree** pane.

Table 8. IP tree pane - icons and color scheme

Display	Peripheral status
CAN1	The peripheral is not configured (no mode is set) and all modes are available.
ADC1	The peripheral is configured (at least one mode is set) and all other modes are available
⚠ ADC3	The peripheral is configured (one mode is set) and at least one of its other modes is unavailable.
⚠ ADC2	The peripheral is not configured (no mode is set) and at least one of its modes is unavailable.
✖ ETH	The peripheral is not configured (no mode is set) and no mode is available. Move the mouse over the IP name to display the tooltip describing the conflict.
CAN1 Mode Disable	Available peripheral mode configurations are shown in plain black.
 DCMI DCMI Disable Disable Slave-8-bits-Embedded-Synchro Slave-8-bits-External-Synchro Slave-10-bits-External-Synchro Slave-12-bits-External-Synchro Slave-14-bits-External-Synchro	The warning yellow icon indicates that at least one mode configuration is no longer available.
✖ ETH Mode Disable	When no more configurations are left for a given peripheral mode, this peripheral is highlighted in red.
 LWIP: LightWeight TCP/IP Not available: Active only with ETH IP configured	Some modes depends on the configuration of other peripherals or middleware modes. A tooltip explains the dependencies when the conditions are not fulfilled.

4.11.2 Chip view

The **Chip** view shows, for the selected part number:

- The MCU in a specific package (BGA, LQFP…)
- The graphical representation of its pinout, each pin being represented with its name (e.g. PC4: pin 4 of GPIO port C) and its current function assignment (e.g. ETH_MII_RXD0) (see *Figure 46* for an example).

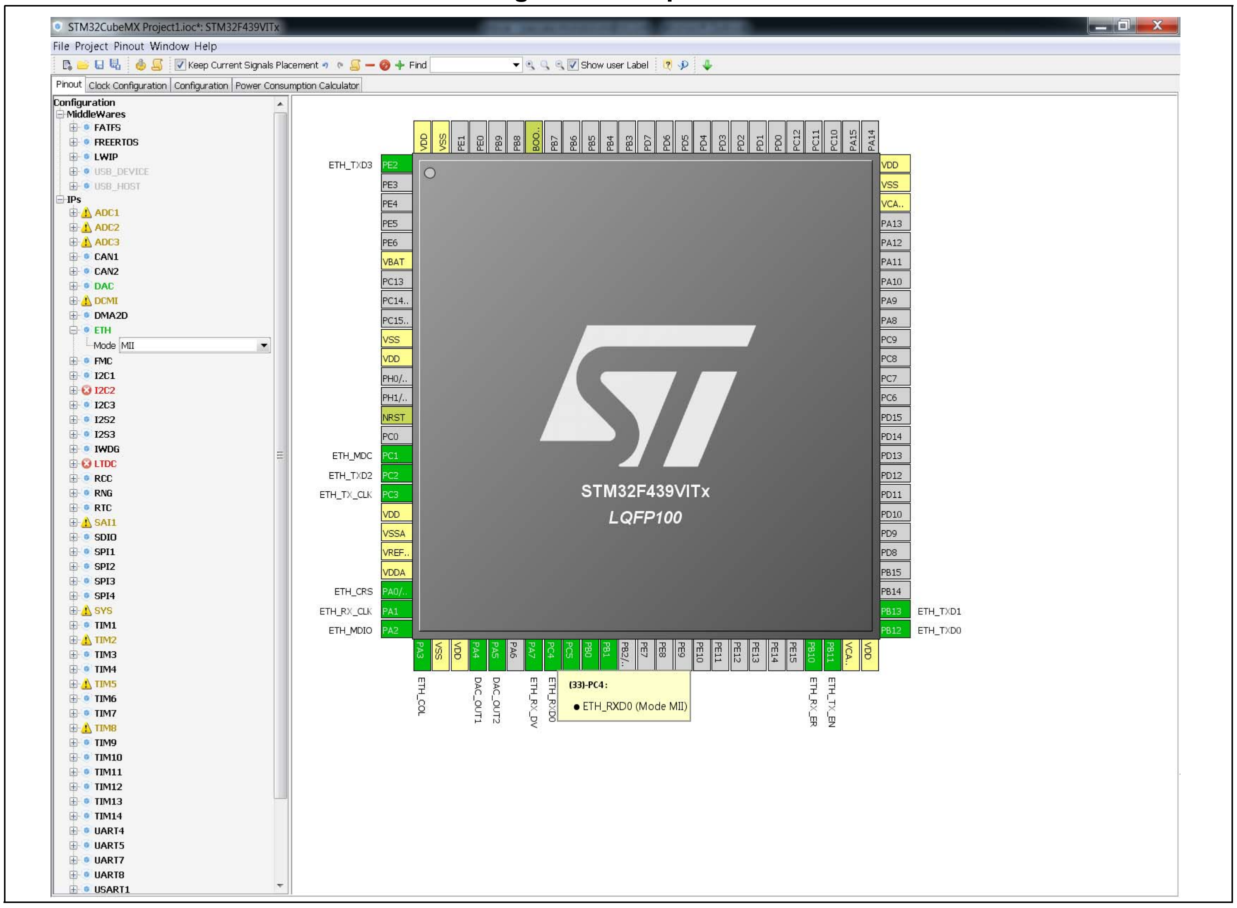
The **Chip** view is automatically refreshed to match the user configuration performed via the peripheral tree. It shows the pins current configuration state.

Assigning pins through the **Chip** view instead of the peripheral pane requires a good knowledge of the MCU since each individual pin can be assigned to a specific function.

Tips and tricks

- Use the mouse wheel to zoom in and out.
- Click and drag the chip diagram to move it. Click **best fit** to reset it to best suited position and size (see *Table 5*).
- Use **Pinout > Generic CSV pinout text file** to export the pinout configuration into text format.
- Some basic controls**,** such as insuring blocks of pins consistency, are built-in. See *Appendix A: STM32CubeMX pin assignment rules* for details.

Figure 46. Chip view



Icons and color schemes

Table 9 shows the icons and color scheme used in the **Chip** view.

Table 9. STM32CubeMX Chip view - Icons and color scheme

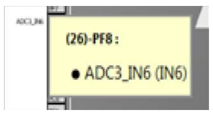
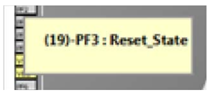
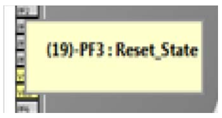
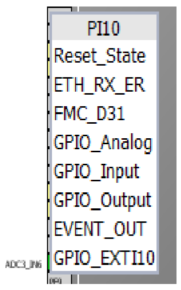
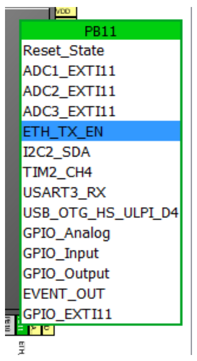
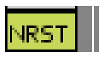
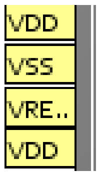
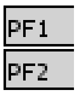
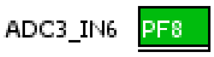
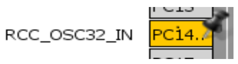
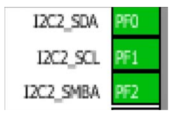
Display	Pin information
  	**Tooltip** indicates the selected pin current configuration: alternate function name, Reset state or GPIO mode. Move your mouse over the pin name to display it. When a pin features alternate pins corresponding to the function currently selected, a popup message prompts the user to perform a CTRL + click to display them. The alternate pins available are highlighted in blue.
 PI10 Reset_State ETH_RX_ER FMC_D31 GPIO_Analog GPIO_Input GPIO_Output EVENT_OUT GPIO_EXTI10	**List** of alternate functions that can be selected for a given pin. By default, no alternate function is configured (pin in reset state). Click the pin name to display the list.
 PB11 Reset_State ADC1_EXTI11 ADC2_EXTI11 ADC3_EXTI11 ETH_TX_EN I2C2_SDA TIM2_CH4 USART3_RX USB_OTG_HS_ULPI_D4 GPIO_Analog GPIO_Input GPIO_Output EVENT_OUT GPIO_EXTI11	When a function has been mapped to the pin, it is highlighted in blue. When it corresponds to a well configured peripheral mode, the list caption is shown in green.
	Boot and reset pins are highlighted in khaki. Their configuration cannot be changed.

Table 9. STM32CubeMX Chip view - Icons and color scheme (continued)

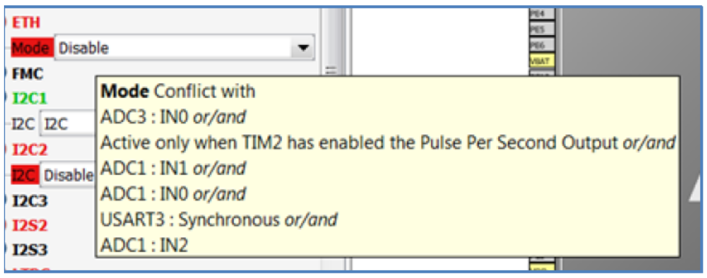
Display	Pin information
	Power dedicated pins are highlighted in yellow. Their configuration cannot be changed.
	Non-configured pins are shown in gray (default state).
	When a signal assignment corresponds to a peripheral mode without ambiguity, the pin color switches to green.
	When the signal assignment does not correspond to a valid peripheral mode configuration, the pin is shown in orange. Additional pins need to be configured to achieve a valid mode configuration.
	When a signal assignment corresponds to a peripheral mode without ambiguity, the pins are shown in green. As an example, assigning the PF2 pin to the I2C2_SMBA signal matches to I2C2 mode without ambiguity and STM32CubeMX configures automatically the other pins (PF0 and PF1) to complete the pin mode configuration.

Tooltips

Move the mouse over IPs and IP modes that are unavailable or partially available to display the tooltips describing the source of the conflict that is which pins are being used by which peripherals.

As an example (see *Figure 47*), the Ethernet (ETH) peripheral is no longer available because there is no possible mode configuration left. A tooltip indicates to which signal are assigned the pins required for this mode (ADC1-IN0 signal, USART3 synchronous signal, etc...).

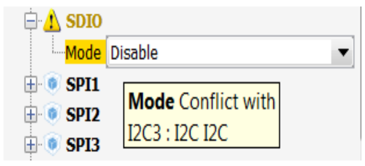
Figure 47. Red highlights and tooltip example: no mode configuration available



Mode Conflict with
ADC3 : IN0 *or/and*
Active only when TIM2 has enabled the Pulse Per Second Output *or/and*
ADC1 : IN1 *or/and*
ADC1 : IN0 *or/and*
USART3 : Synchronous *or/and*
ADC1 : IN2

In the next example (see *Figure 48*), the SDIO peripheral is partially available because at least one of its modes is unavailable: the necessary pins are already assigned to the I2C mode of the I2C3 peripheral.

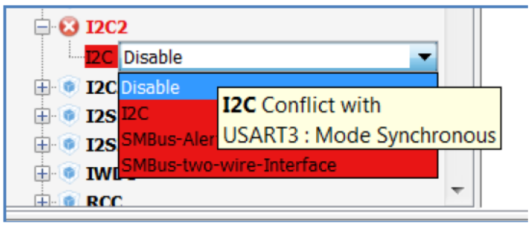
Figure 48. Orange highlight and tooltip example: some configurations unavailable



Mode Conflict with
I2C3 : I2C I2C

In this last example (see *Figure 49*) I2C2 peripheral is unavailable because there is no mode function available. A tooltip shows for each function where all the remapped pins have been allocated (USART3 synchronous mode).

Figure 49. Tooltip example: all configurations unavailable



I2C Conflict with
USART3 : Mode Synchronous

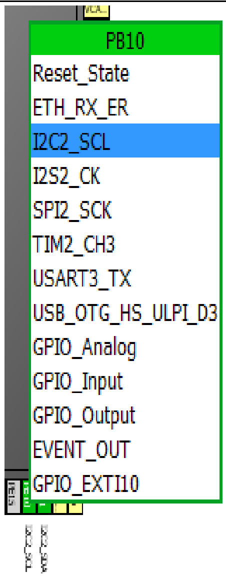
4.11.3 Chip view advanced actions

Manually modifying pin assignments

To manually modify a pin assignment, follow the sequence below:

1. Click the pin in the **Chip** view to display the list of all other possible alternate functions together with the current assignment highlighted in blue (see *Figure 50*).
2. Click to select the new function to assign to the pin.

Figure 50. Modifying pin assignments from the Chip view



Manually remapping a function to another pin

To manually remap a function to another pin, follow the sequence below:

1. Press the CTRL key and click the pin in the **Chip** view. Possible pins for relocation, if any, are highlighted in blue.
2. Drag the function to the target pin.

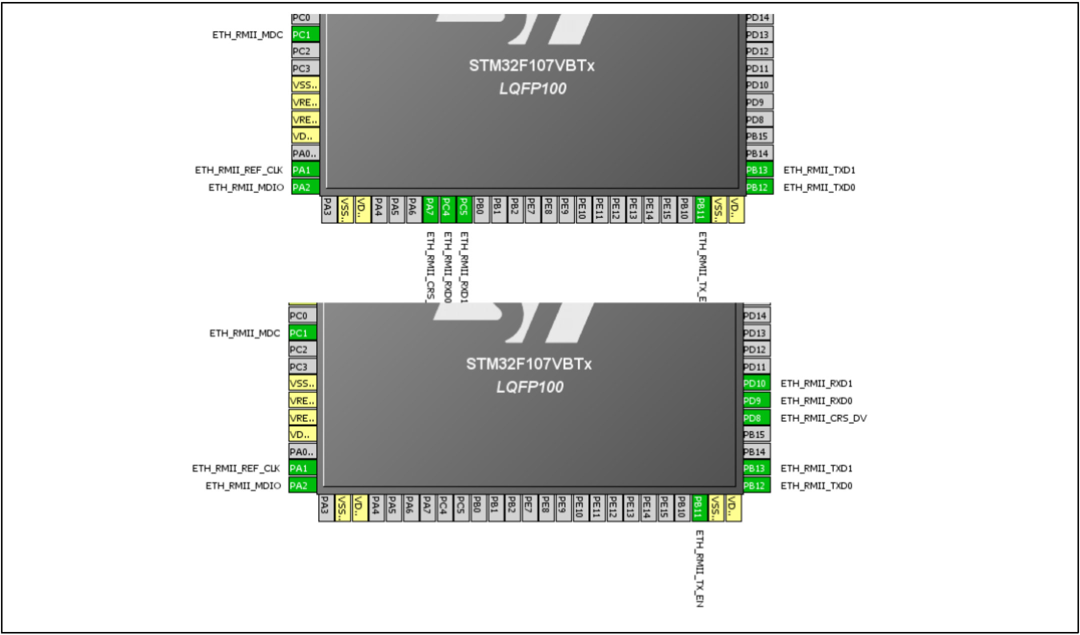
Caution: A pin assignment performed from the Chip view overwrites any previous assignment.

Manual remapping with destination pin ambiguity

For MCUs with block of pins consistency (STM32F100x/ F101x/ F102x/ F103x and STM32F105x/F107x), the destination pin can be ambiguous,e.g. there can be more than one destination block including the destination pin. To display all the possible alternative remapping blocks, move the mouse over the target pin.

Note: A "block of pins" is a group of pins that must be assigned together to achieve a given peripheral mode. As shown in Figure 51, two blocks of pins are available on a STM32F107xx MCU to configure the Ethernet Peripheral in RMII synchronous mode: {PC1, PA1, PA2, PA7, PC4, PC5, PB11, PB12, PB13, PB5} and {PC1, PA1, PA2, PD10, PD9, PD8, PB11, PB12, PB13, PB5}.

Figure 51. Example of remapping in case of block of pins consistency



Resolving pin conflicts

To resolve the pin conflicts that may occur when some peripheral modes use the same pins, STM32CubeMX attempts to reassign the peripheral mode functions to other pins. The peripherals for which pin conflicts could not be solved are highlighted in red or orange with a tooltip describing the conflict.

If the conflict cannot be solved by remapping the modes, the user can try the following:

- If the ☑ Keep Current Signals Placement box is checked, try to select the peripherals in a different sequence.
- Uncheck the **Keep Current Signals Placement** box and let STM32CubeMX try all the remap combinations to find a solution.
- **Manually remap** a mode of a peripheral when you cannot use it because there is no pin available for one of the signals of that mode.

4.11.4 Keep Current Signals Placement

This checkbox is available from the toolbar when the **Pinout** view is selected (see *Figure 26* and *Table 5*). It can be selected or unselected at any time during the configuration. It is unselected by default.

It is recommended to keep the checkbox unchecked for an optimized placement of the peripherals (maximum number of peripherals concurrently used).

The **Keep Current Signals Placement** checkbox should be selected when the objective is to match a board design.

Keep Current Signals Placement is unchecked

This allows STM32CubeMX to remap previously mapped blocks to other pins in order to serve a new request (selection of a new IP mode or a new IP mode function) which conflicts with the current pinout configuration.

Keep Current Signals Placement is checked

This ensures that all the functions corresponding to a given peripheral mode remain allocated (mapped) to a given pin. Once the allocation is done, STM32CubeMX cannot move a peripheral mode function from one pin to another. New configuration requests are served if it is feasible within current pin configuration.

This functionality is useful to:

- Lock all the pins corresponding to peripherals that have been configured using the **Peripherals** panel.
- Maintain a function mapped to a pin while doing manual remapping from the **Chip** view.

Tip

If a mode becomes unavailable (highlighted in red), try to find another pin remapping configuration for this mode by following the steps below:

1. From the **Chip** view, unselect the assigned functions one by one until the mode becomes available again.
2. Then, select the mode again and continue the pinout configuration with the new sequence (see *Appendix A: STM32CubeMX pin assignment rules* for a remapping example). This operation being time consuming, it is recommended to unselect the **Keep Current Signals Placement** checkbox.

Note: Even if Keep Current Signals placement is unchecked, GPIO_ functions (excepted GPIO_EXTI functions) are not moved by STM32CubeMX.

4.11.5 Pinning and labeling signals on pins

STM32CubeMX comes with a feature allowing the user to selectively lock (or pin) signals to pins: This will prevent STM32CubeMX from automatically moving the pinned signals to other pins when resolving conflicts.There is also the possibility to label the signals: User labels are used for code generation (see Section 5.1 for details).

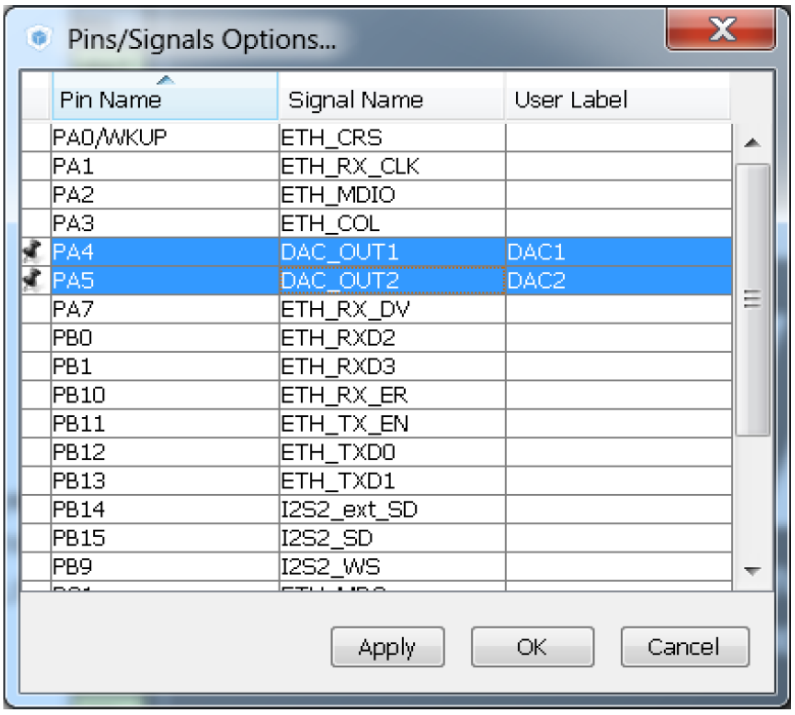
STM32CubeMX comes with a feature allowing the user to selectively lock (or pin) signals to pins. This prevents STM32CubeMX from automatically moving pinned signals to other pins when resolving conflicts. Labels, that are used for code generation, can also be assigned to the signals (see *Section 5.1* for details).

There are several ways to pin, unpin and label the signals:

1. From the **Chip** view, right-click a pin with a signal assignment. This opens a contextual menu:
   a) For unpinned signals, select **Signal Pinning** to pin the signal. A pin icon is then displayed on the relevant pin. The signal can no longer be moved automatically (for example when resolving pin assignment conflicts).
   b) For pinned signals, select **Signal Unpinning** to unpin the signal. The pin icon is removed. From now on, to resolve a conflict (such as peripheral mode conflict), this signal can be moved to another pin, provided the Keep user placement option is unchecked.
   c) Select **Enter User Label** to specify a user defined label for this signal. The new label will replacing the default signal name in the **Chip** view.
2. From the pinout menu, select **Pins/Signals Options**

   The Pins/Signals Options window (see *Figure 52*) lists all configured pins.
   a) Click the first column to individually pin/unpin signals.
   b) Select multiple rows and right-click to open the contextual menu and select Signal(s) Pinning or Unpinning.

Figure 52. Pins/Signals Options window



Pin Name	Signal Name	User Label
PA0/WKUP	ETH_CRS	
PA1	ETH_RX_CLK	
PA2	ETH_MDIO	
PA3	ETH_COL	
PA4	DAC_OUT1	DAC1
PA5	DAC_OUT2	DAC2
PA7	ETH_RX_DV	
PB0	ETH_RXD2	
PB1	ETH_RXD3	
PB10	ETH_RX_ER	
PB11	ETH_TX_EN	
PB12	ETH_TXD0	
PB13	ETH_TXD1	
PB14	I2S2_ext_SD	
PB15	I2S2_SD	
PB9	I2S2_WS	

c) Select the User Label field to edit the field and enter a user-defined label.

d) Order list alphabetically by Pin or Signal name by clicking the column header. Click once more to go back to default i.e. to list ordered according to pin placement on MCU.

Note: Even if a signal is pinned, it is still possible however to manually change the pin signal assignment from the ***Chip*** *view: click the pin to display other possible signals for this pin and select the relevant one.*

4.11.6 Setting HAL timebase source

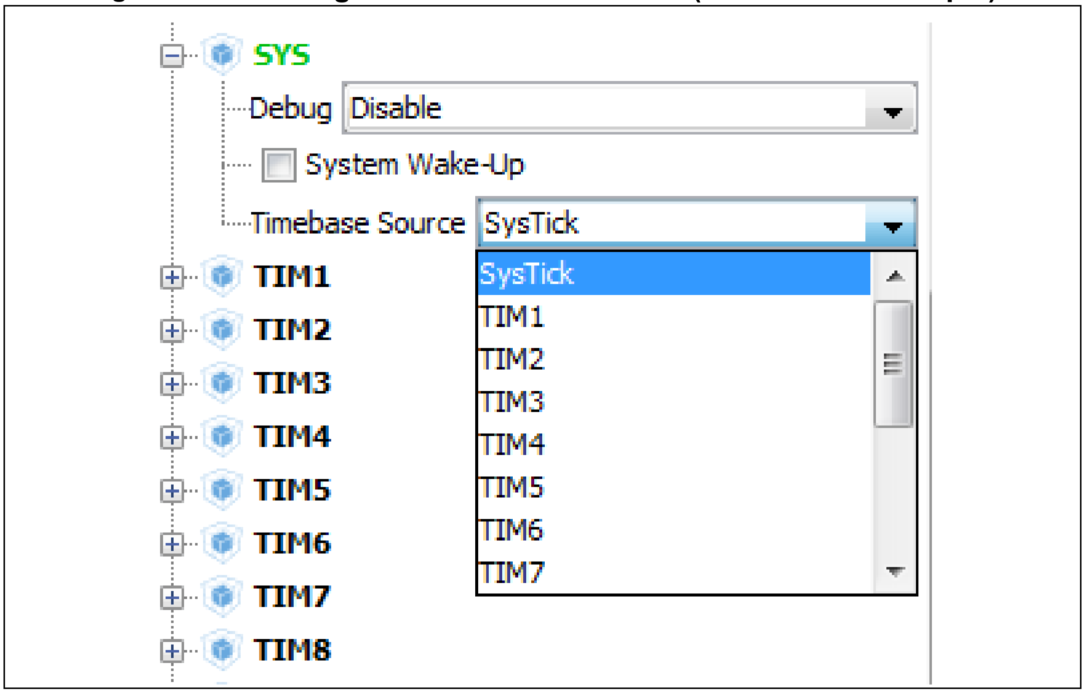
By default, the STM32Cube HAL is built around a unique timebase source which is the ARM-Cortex system timer (SysTick).

However, HAL-timebase related functions are defined as weak so that they can be overloaded to use another hardware timebase source. This is strongly recommended when the application uses an RTOS, since this middleware has full control on the SysTick configuration (tick and priority) and most RTOSs force the SysTick priority to be the lowest.

Using the SysTick remains acceptable if the application respects the HAL programming model, that is, does not perform any call to HAL timebase services within an Interrupt Service Request context (no dead lock issue).

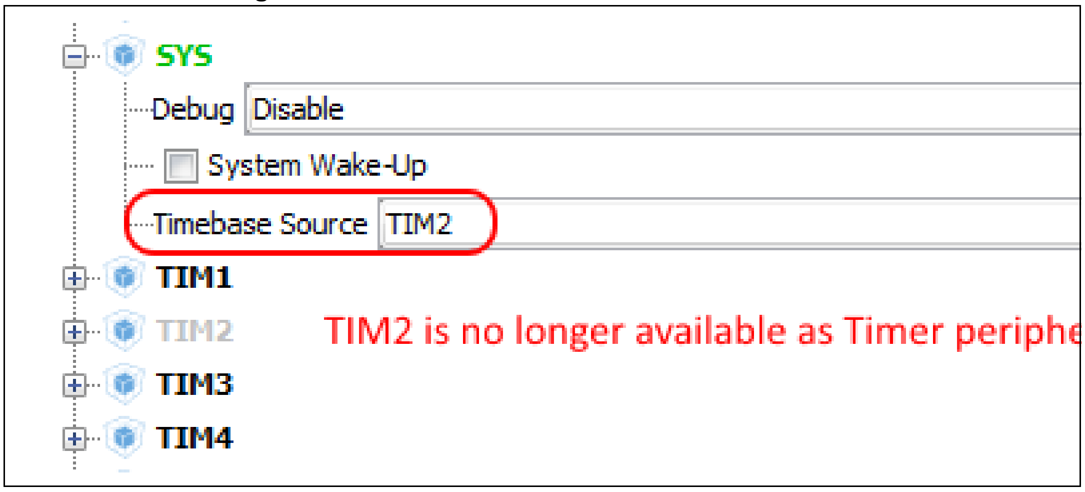
To change the HAL timebase source, go to the SYS peripheral in the **IP tree** pane and select a clock among the available clock sources: SysTick, TIM1, TIM2,... (see *Figure 53*).

Figure 53. Selecting a HAL timebase source (STM32F407 example)


SYS
Debug Disable
System Wake-Up
Timebase Source SysTick

When used as timebase source, a given peripheral is grayed and can no longer be selected (see *Figure 54*).

Figure 54. TIM2 selected as HAL timebase source

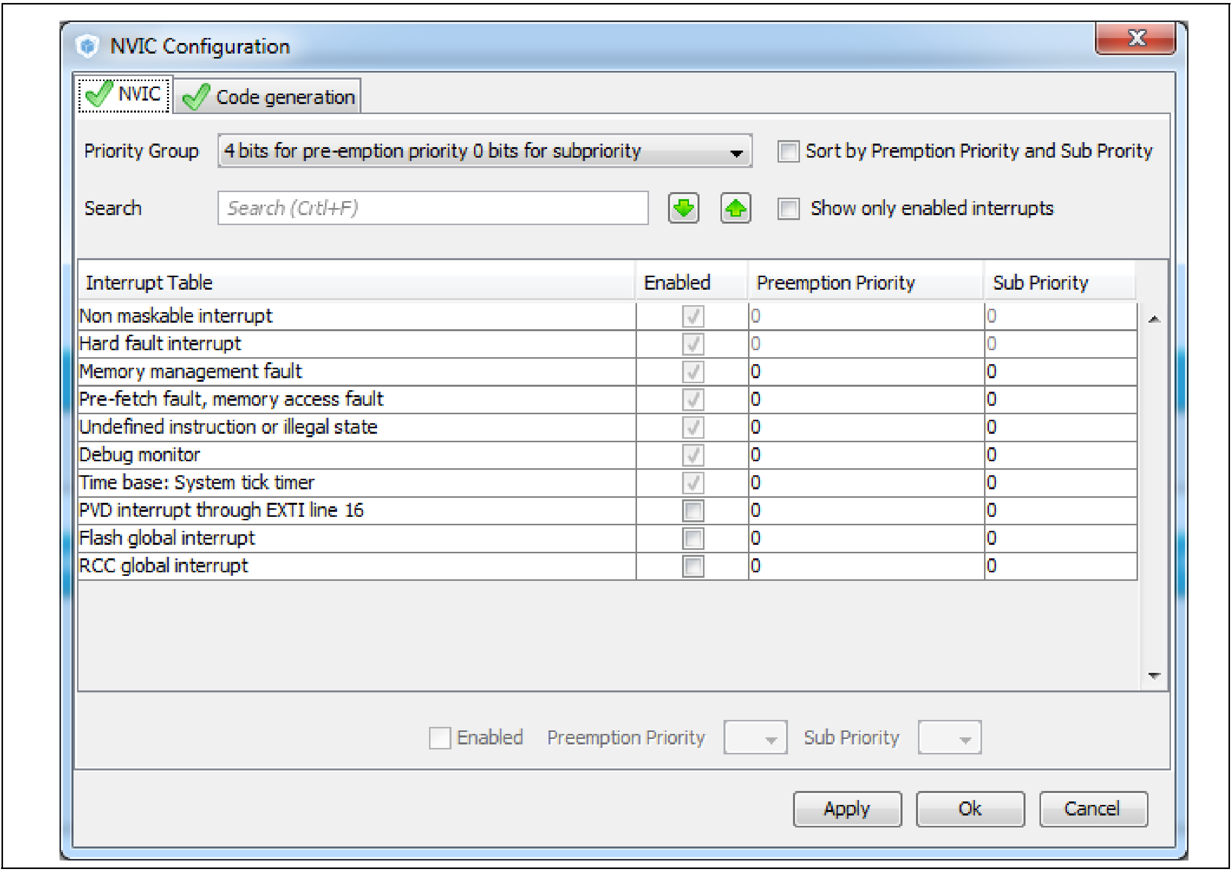

SYS
Debug Disable
System Wake-Up
Timebase Source TIM2

As illustrated in the following examples, the selection of the HAL timebase source and the use of FreeRTOS influence the generated code.

Example of configuration using SysTick without FreeRTOS

As illustrated in *Figure 55*, the Systick priority is set to 0 (High) when using the Systick without FreeRTOS.

Figure 55. NVIC settings when using systick as HAL timebase, no FreeRTOS



Priority Group: 4 bits for pre-emption priority 0 bits for subpriority
☐ Sort by Premption Priority and Sub Prority
☐ Show only enabled interrupts

Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	✓	0	0
Hard fault interrupt	✓	0	0
Memory management fault	✓	0	0
Pre-fetch fault, memory access fault	✓	0	0
Undefined instruction or illegal state	✓	0	0
Debug monitor	✓	0	0
Time base: System tick timer	✓	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0

☐ Enabled Preemption Priority Sub Priority

Interrupt priorities (in main.c) and handler code (in stm32f4xx_it.c) are generated accordingly:

- main.c file

```
/* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
```

- stm32f4xx_it.c file

```
/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */
```

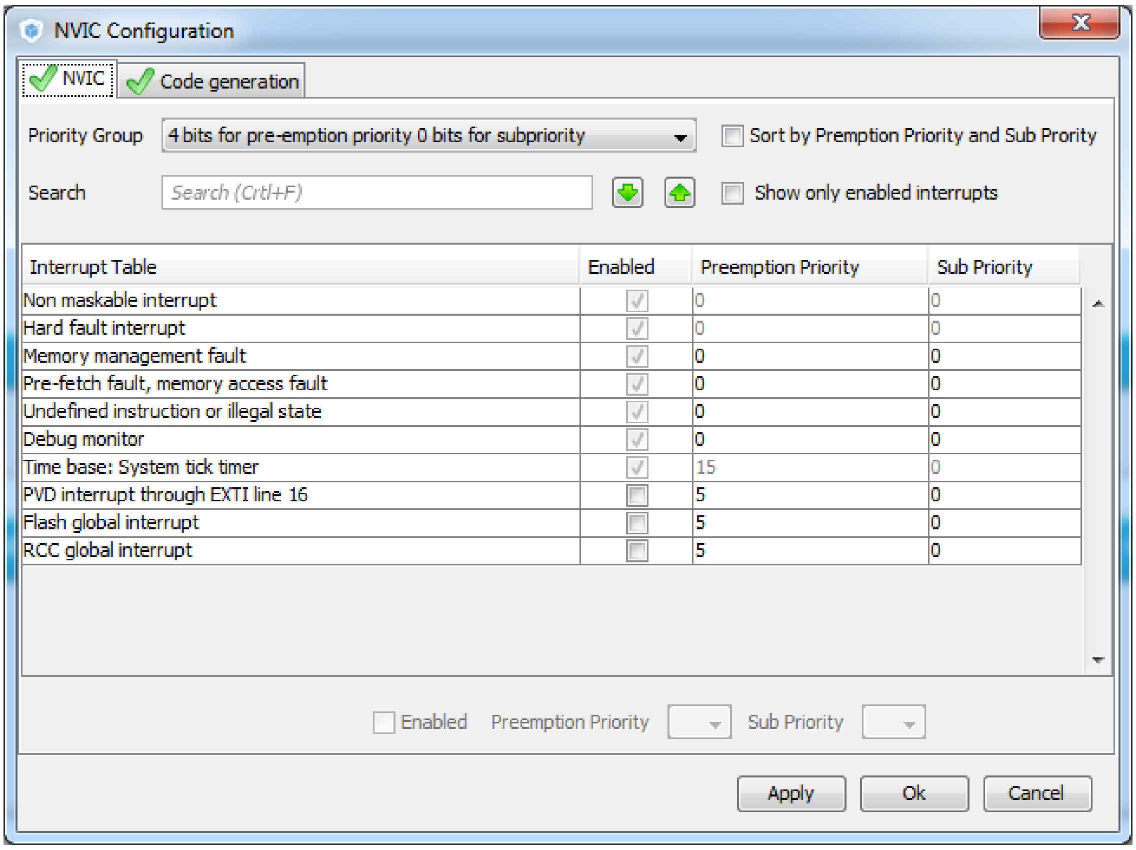
```
    /* USER CODE END SysTick_IRQn 0 */
     HAL_IncTick();
     HAL_SYSTICK_IRQHandler();
     /* USER CODE BEGIN SysTick_IRQn 1 */

     /* USER CODE END SysTick_IRQn 1 */
  }
```

Example of configuration using SysTick and FreeRTOS

As illustrated in *Figure 56*, the Systick priority is set to 15 (Low) when using the SysTick with FreeRTOS.

Figure 56. NVIC settings when using FreeRTOS and SysTick as HAL timebase



Priority Group: 4 bits for pre-emption priority 0 bits for subpriority

Sort by Premption Priority and Sub Prority

Show only enabled interrupts

Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	✓	0	0
Hard fault interrupt	✓	0	0
Memory management fault	✓	0	0
Pre-fetch fault, memory access fault	✓	0	0
Undefined instruction or illegal state	✓	0	0
Debug monitor	✓	0	0
Time base: System tick timer	✓	15	0
PVD interrupt through EXTI line 16	☐	5	0
Flash global interrupt	☐	5	0
RCC global interrupt	☐	5	0