5.2 Custom code generation

STM32CubeMX supports custom code generation by means of a FreeMarker template engine (see http://www.freemarker.org).

5.2.1 STM32CubeMX data model for FreeMarker user templates

STM32CubeMX can generate a custom code based on a FreeMarker template file (.ftl extension) for any of the following MCU configuration information:

- List of MCU peripherals used by the user configuration
- List of parameters values for those peripherals
- List of resources used by these peripherals: GPIO, DMA requests and interrupts.

The user template file must be compatible with STM32CubeMX data model. This means that the template must start with the following lines:

```
[#ftl]
[#list configs as dt]
[#assign data = dt]
[#assign peripheralParams =dt.peripheralParams]
[#assign peripheralGPIOParams =dt.peripheralGPIOParams]
[#assign usedIPs =dt.usedIPs]
```

and end with

```
[/#list]
```

A sample template file is provided for guidance (see *Figure 112: extra_templates folder – default content*).

STM32CubeMX will also generate user-specific code if any is available within the template.

As shown in the below example, when the sample template is used, the ftl commands are provided as comments next to the data they have generated:

FreeMarker command in template:

```
${peripheralParams.get("RCC").get("LSI_VALUE")}
```

Resulting generated code:

```
LSI_VALUE :  32000       [peripheralParams.get("RCC").get("LSI_VALUE")]
```

5.2.2 Saving and selecting user templates

The user can either place the FreeMarker template files under STM32CubeMX installation path within the db/extra_templates folder or in any other folder.

Then for a given project, the user will select the template files relevant for his project via the **Template Settings** window accessible from the **Project Settings** menu (see *Section 4.8: Project Settings window*)

5.2.3 Custom code generation

To generate custom code, the user must place the FreeMarker template file under STM32CubeMX installation path within the db/extra_templates folder (see *Figure 113:*

extra_templates folder with user templates).

The template filename must follow the naming convention <user filename>_<file extension>.ftl in order to generate the corresponding custom file as <user filename>.<file extension>.

By default, the custom file is generated in the user project root folder, next to the .ioc file (see *Figure 114: Project root folder with corresponding custom generated files*).

To generate the custom code in a different folder, the user shall match the destination folder tree structure in the extra_template folder (see *Figure 115: User custom folder for templates*).

Figure 112. extra_templates folder – default content

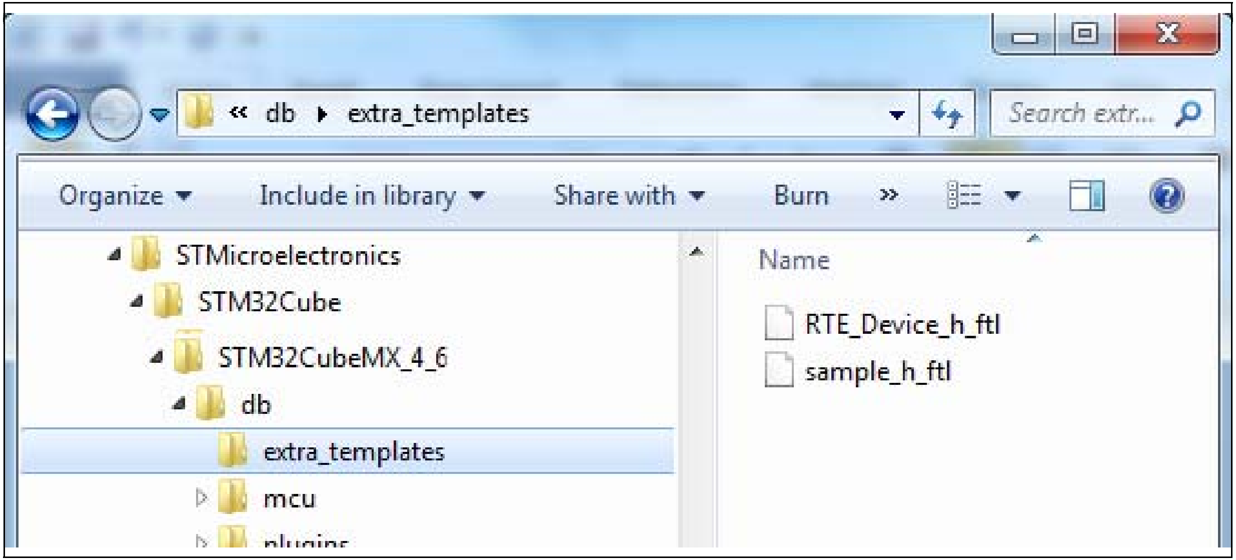


Figure 113. extra_templates folder with user templates

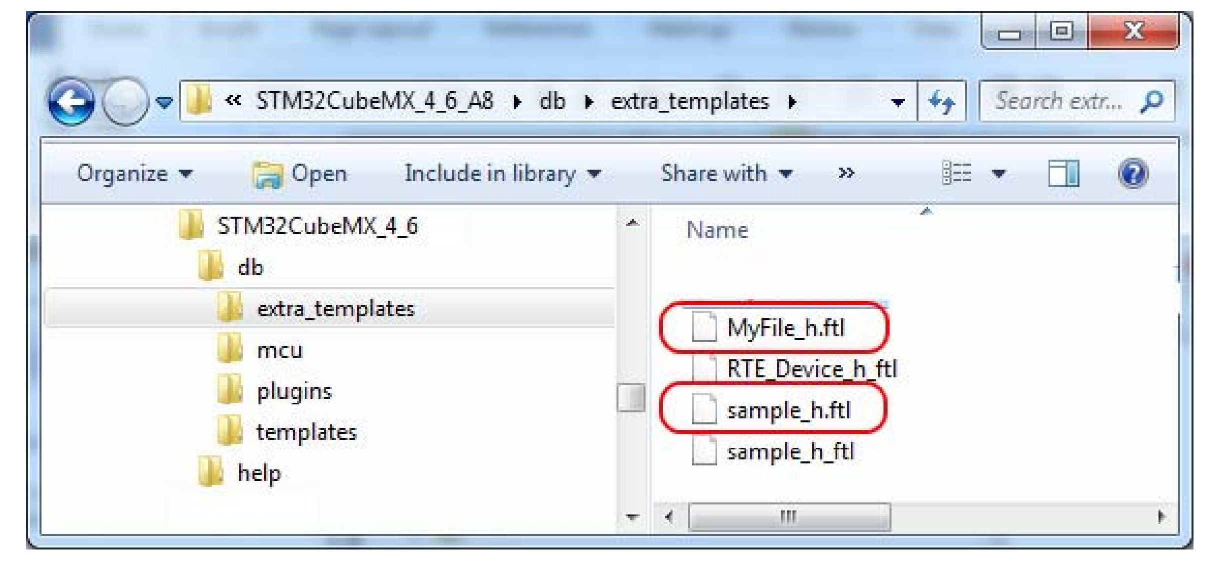


Figure 114. Project root folder with corresponding custom generated files

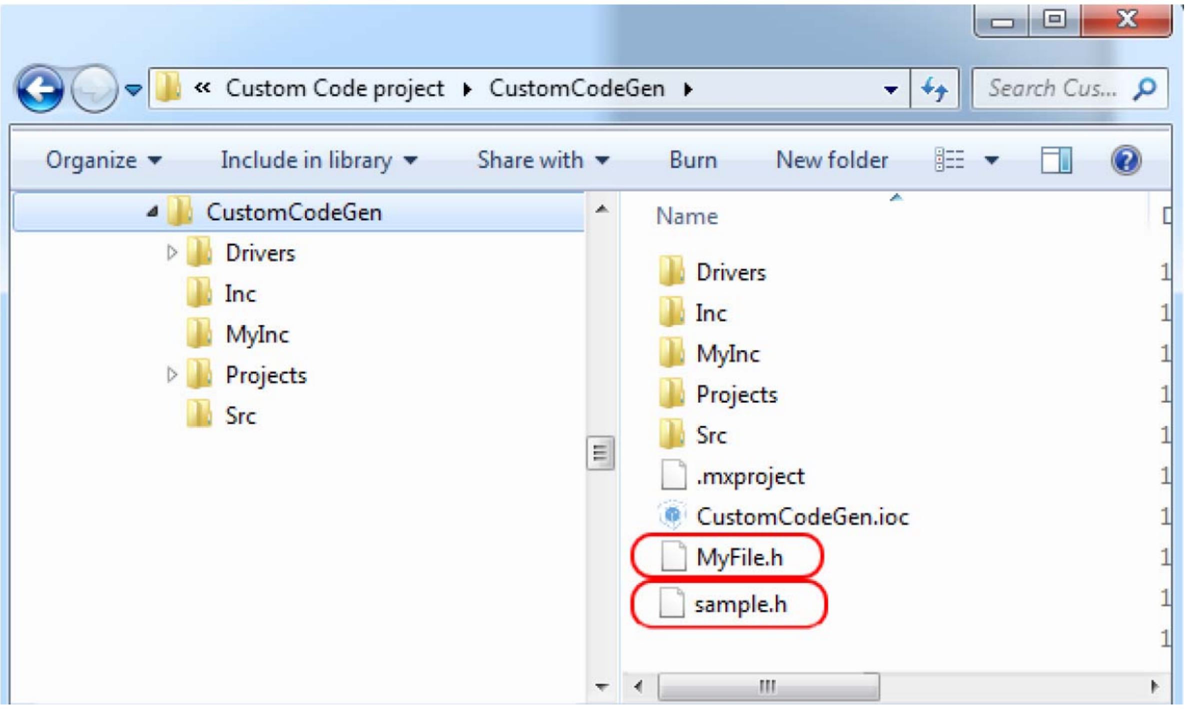


Figure 115. User custom folder for templates

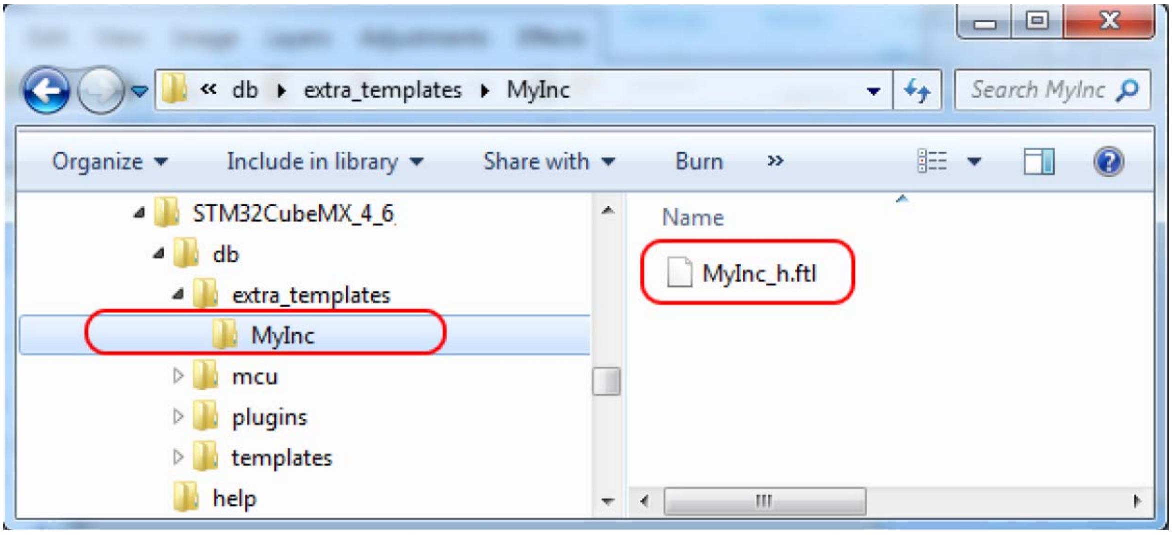
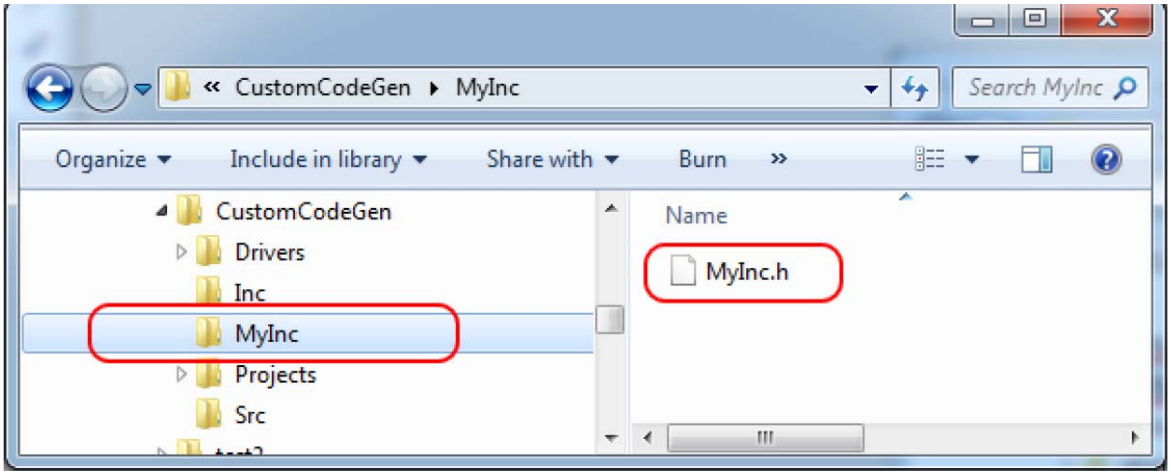


Figure 116. Custom folder with corresponding custom generated files



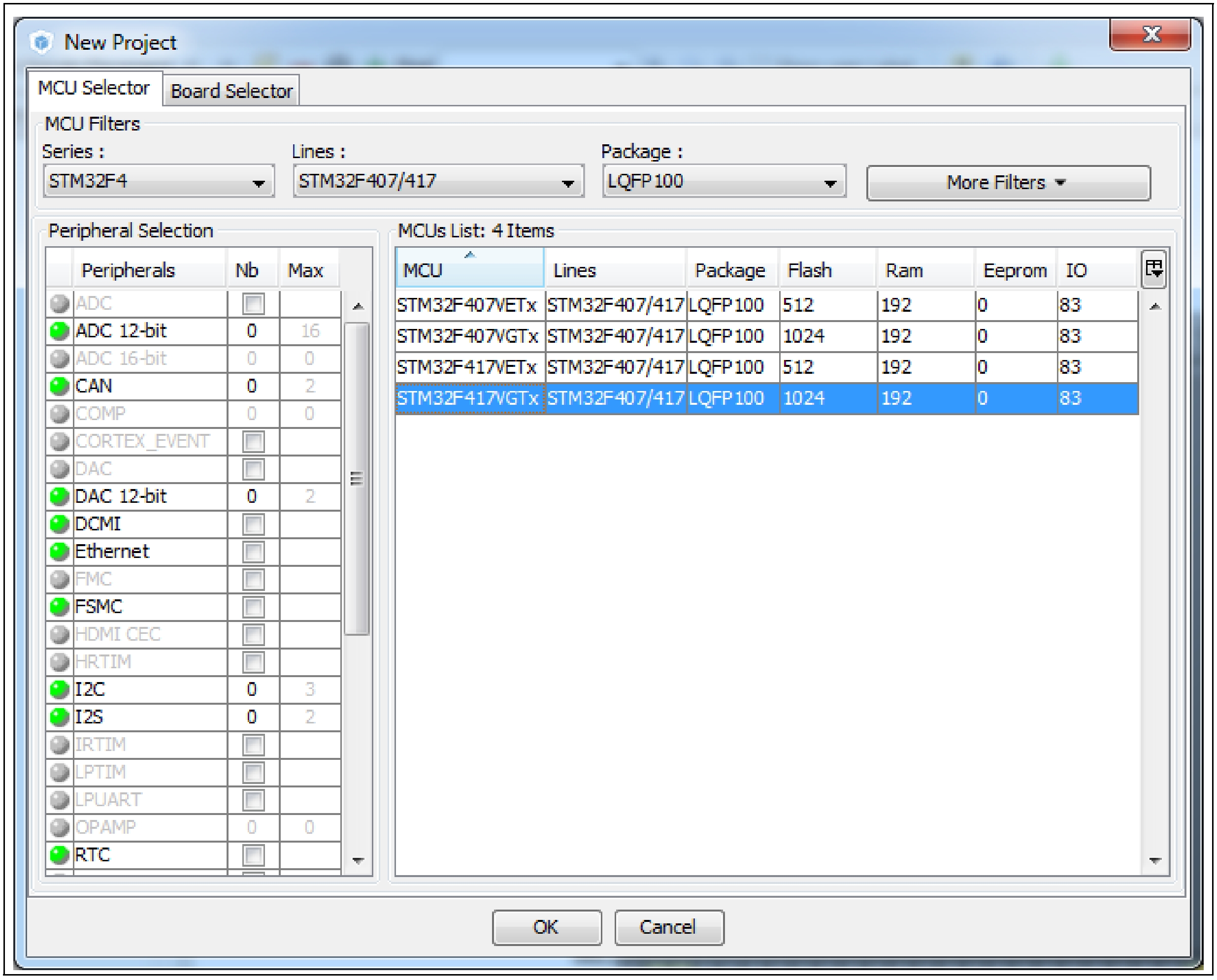
6 Tutorial 1: From pinout to project C code generation using an STM32F4 MCU

This section describes the configuration and C code generation process. It takes as an example a simple LED toggling application running on the STM32F4DISCOVERY board.

6.1 Creating a new STM32CubeMX Project

1. Select **File > New project** from the main menu bar or **New project** from the Welcome page.
2. Select the MCU Selector tab and filter down the STM32 portfolio by selecting STM32F4 as 'Series', STM32F407 as 'Lines', and LQFP100 as 'Package' (see *Figure 117*).
3. Select the STM32F407VGTx from the MCU list and click OK.

Figure 117. MCU selection



Series : STM32F4

Lines : STM32F407/417

Package : LQFP100

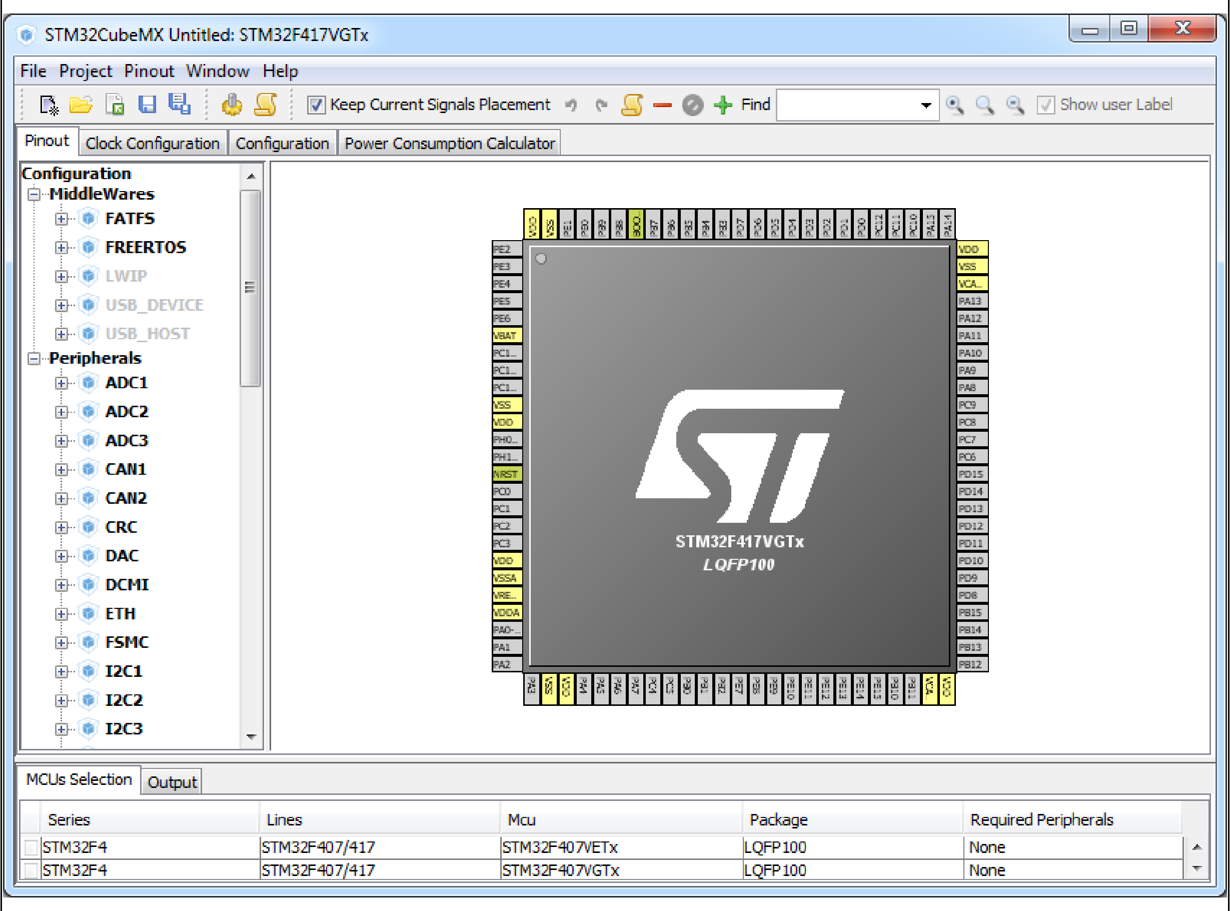
MCUs List: 4 Items

Peripherals	Nb	Max
ADC	☐	
ADC 12-bit	0	16
ADC 16-bit	0	0
CAN	0	2
COMP	0	0
CORTEX_EVENT	☐	
DAC	☐	
DAC 12-bit	0	2
DCMI	☐	
Ethernet	☐	
FMC	☐	
FSMC	☐	
HDMI CEC	☐	
HRTIM	☐	
I2C	0	3
I2S	0	2
IRTIM	☐	
LPTIM	☐	
LPUART	☐	
OPAMP	0	0
RTC	☐	

MCU	Lines	Package	Flash	Ram	Eeprom	IO
STM32F407VETx	STM32F407/417	LQFP100	512	192	0	83
STM32F407VGTx	STM32F407/417	LQFP100	1024	192	0	83
STM32F417VETx	STM32F407/417	LQFP100	512	192	0	83
STM32F417VGTx	STM32F407/417	LQFP100	1024	192	0	83

STM32CubeMX views are then populated with the selected MCU database (see *Figure 118*).

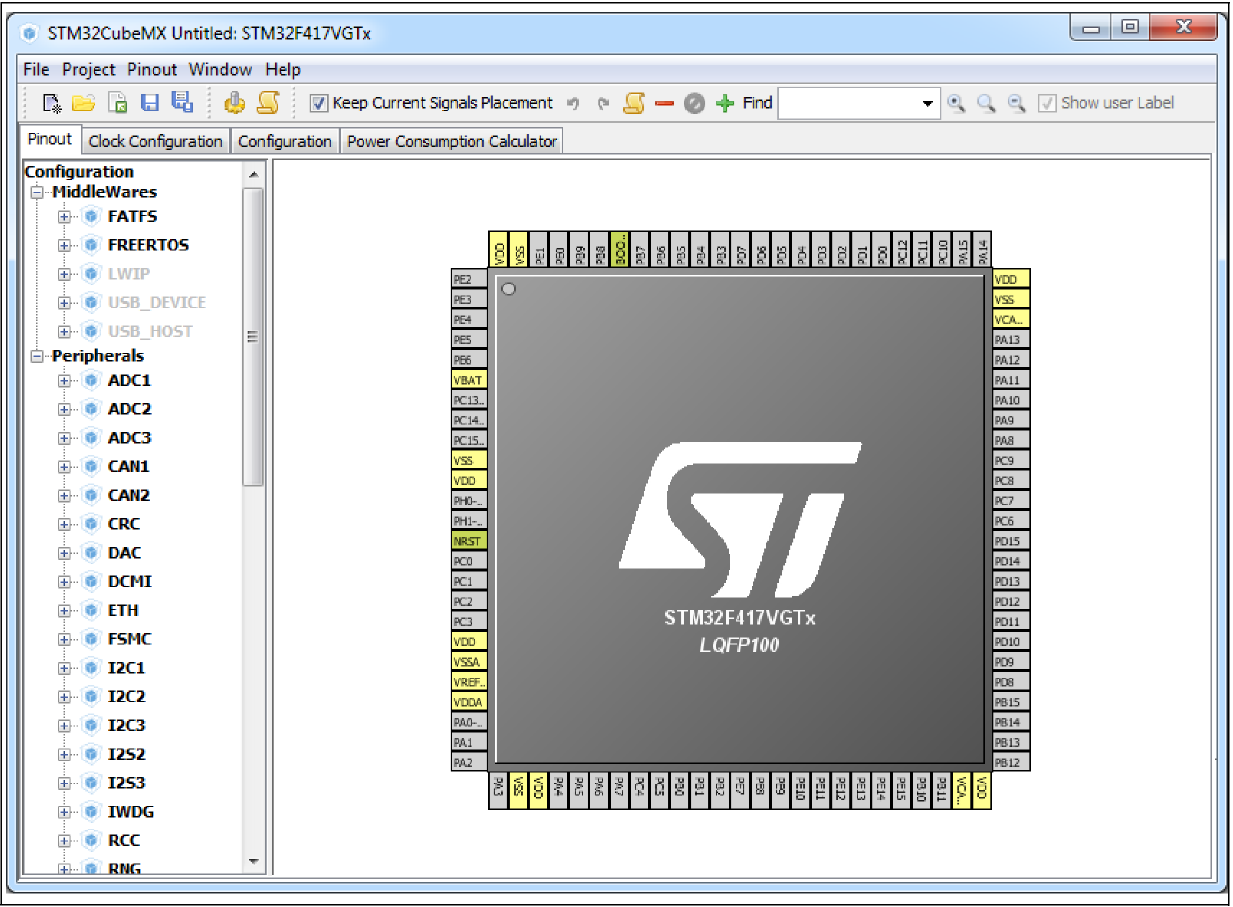
Figure 118. Pinout view with MCUs selection



Series	Lines	Mcu	Package	Required Peripherals
STM32F4	STM32F407/417	STM32F407VETx	LQFP100	None
STM32F4	STM32F407/417	STM32F407VGTx	LQFP100	None

Optionally, remove the MCUs Selection bottom window by unselecting **Window> Outputs** sub-menu (see *Figure 119*).

Figure 119. Pinout view without MCUs selection window


STM32CubeMX Untitled: STM32F417VGTx
File Project Pinout Window Help
☑ Keep Current Signals Placement
Find
☑ Show user Label
Pinout
Clock Configuration
Configuration
Power Consumption Calculator
Configuration
MiddleWares
FATFS
FREERTOS
LWIP
USB_DEVICE
USB_HOST
Peripherals
ADC1
ADC2
ADC3
CAN1
CAN2
CRC
DAC
DCMI
ETH
FSMC
I2C1
I2C2
I2C3
I2S2
I2S3
IWDG
RCC
RNG
STM32F417VGTx
LQFP100
VDD VSS PE1 PE0 PB9 PB8 BOO.. PB7 PB6 PB5 PB4 PB3 PD7 PD6 PD5 PD4 PD3 PD2 PD1 PD0 PC12 PC11 PC10 PA15 PA14
PE2 PE3 PE4 PE5 PE6 VBAT PC13.. PC14.. PC15.. VSS VDD PH0-.. PH1-.. NRST PC0 PC1 PC2 PC3 VDD VSSA VREF.. VDDA PA0-.. PA1 PA2
VDD VSS VCA.. PA13 PA12 PA11 PA10 PA9 PA8 PC9 PC8 PC7 PC6 PD15 PD14 PD13 PD12 PD11 PD10 PD9 PD8 PB15 PB14 PB13 PB12
PA3 VSS VDD PA4 PA5 PA6 PA7 PC4 PC5 PB0 PB1 PB2 PE7 PE8 PE9 PE10 PE11 PE12 PE13 PE14 PE15 PB10 PB11 VCA.. VDD

6.2 Configuring the MCU pinout

For a detailed description of menus, advanced actions and conflict resolutions, refer to *Section 4: STM32CubeMX User Interface* and *Appendix A: STM32CubeMX pin assignment rules*.

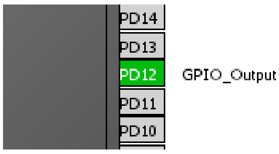
1. By default, STM32CubeMX shows the **Pinout** view.
2. By default, □ Keep Current Signals Placement is unchecked allowing STM32CubeMX to move the peripheral functions around and to find the optimal pin allocation, that is the one that accommodates the maximum number of peripheral modes.

   Since the MCU pin configurations must match the STM32F4DISCOVERY board, enable ☑ Keep Current Signals Placement for STM32CubeMX to maintain the peripheral function allocation (mapping) to a given pin.

   This setting is saved as a user preference in order to be restored when reopening the tool or when loading another project.
3. Select the required peripherals and peripheral modes:

   a) Configure the GPIO to output the signal on the STM32F4DISCOVERY green LED by right-clicking PD12 from the **Chip** view, then select GPIO_output:

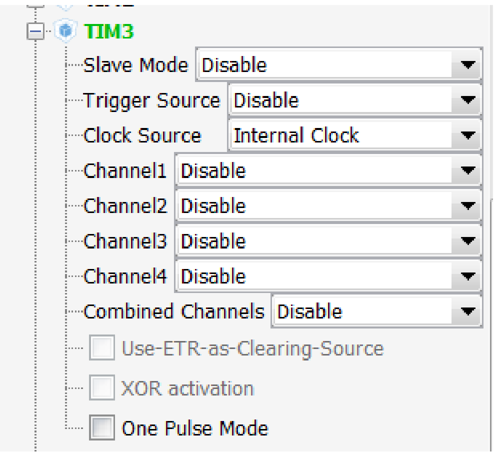
Figure 120. GPIO pin configuration



GPIO_Output

b) Enable a timer to be used as timebase for toggling the LED. This is done by selecting Internal Clock as TIM3 Clock source from the peripheral tree (see *Figure 121*).

Figure 121. Timer configuration



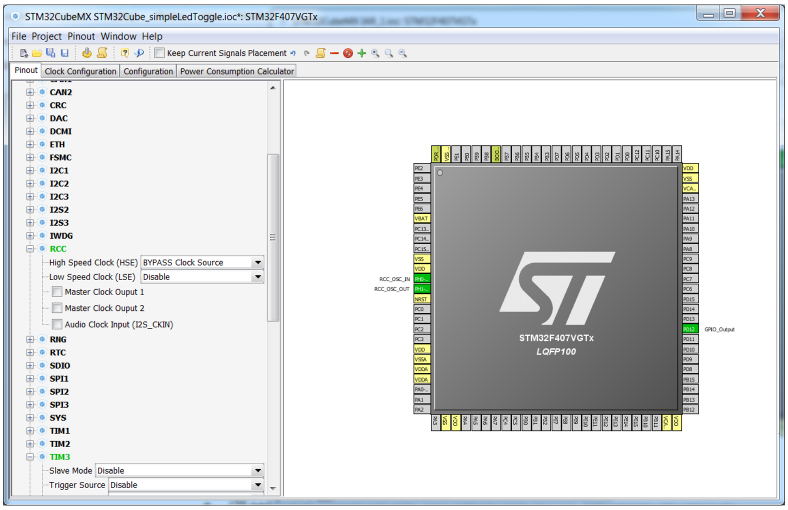
TIM3

- Slave Mode: Disable
- Trigger Source: Disable
- Clock Source: Internal Clock
- Channel1: Disable
- Channel2: Disable
- Channel3: Disable
- Channel4: Disable
- Combined Channels: Disable
- □ Use-ETR-as-Clearing-Source
- □ XOR activation
- □ One Pulse Mode

c) You can also configure the RCC in order to use an external oscillator as potential clock source (see *Figure 122*).

This completes the pinout configuration for this example.

Figure 122. Simple pinout configuration



STM32CubeMX STM32Cube_simpleLedToggle.ioc*: STM32F407VGTx

RCC

High Speed Clock (HSE): BYPASS Clock Source

Low Speed Clock (LSE): Disable

□ Master Clock Ouput 1

□ Master Clock Ouput 2

□ Audio Clock Input (I2S_CKIN)

TIM3

Slave Mode: Disable

Trigger Source: Disable

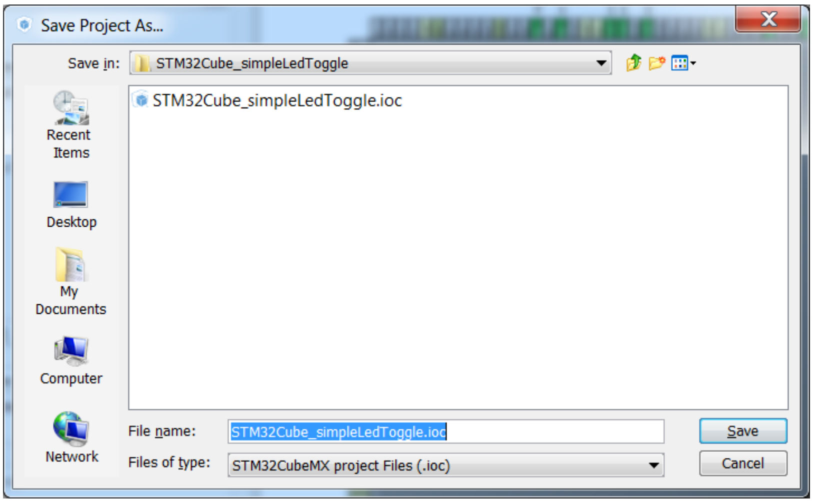
Note: *Starting with STM32CubeMX 4.2, the user can skip the pinout configuration by directly loading ST Discovery board configuration from the Board selector tab.*

6.3 Saving the project

1. Click 💾 to save the project.

When saving for the first time, select a destination folder and filename for the project. The .ioc extension is added automatically to indicate this is an STM32CubeMX configuration file.

Figure 123. Save Project As window



Save Project As...

Save in: STM32Cube_simpleLedToggle

STM32Cube_simpleLedToggle.ioc

File name: STM32Cube_simpleLedToggle.ioc

Files of type: STM32CubeMX project Files (.ioc)

2. Click 🗐 to save the project under a different name or location.

6.4 Generating the report

Reports can be generated at any time during the configuration:

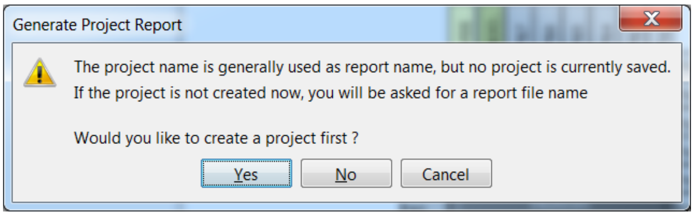
1. Click to generate .pdf and .txt reports.

   If a project file has not been created yet, a warning prompts the user to save the project first and requests a project name and a destination folder (see *Figure 124*). An .ioc file is then generated for the project along with a .pdf and .txt reports with the same name.

   Answering “No” will require to provide a name and location for the report only.

   A confirmation message is displayed when the operation has been successful (see *Figure 125*).

Figure 124. Generate Project Report - New project creation

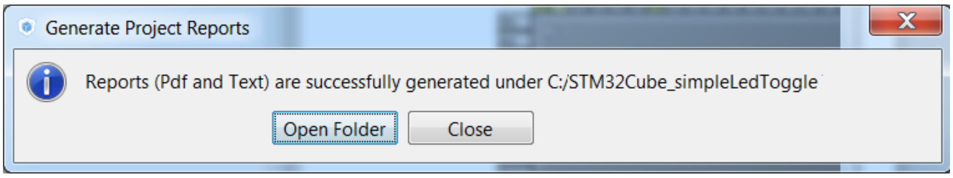


Generate Project Report

The project name is generally used as report name, but no project is currently saved.
If the project is not created now, you will be asked for a report file name

Would you like to create a project first ?

Figure 125. Generate Project Report - Project successfully created



Generate Project Reports

Reports (Pdf and Text) are successfully generated under C:/STM32Cube_simpleLedToggle

2. Open the .pdf report using Adobe Reader or the .txt report using your favorite text editor. The reports summarize all the settings and MCU configuration performed for the project.

6.5 Configuring the MCU Clock tree

The following sequence describes how to configure the clocks required by the application based on an STM32F4 MCU.

STM32CubeMX automatically generates the system, CPU and AHB/APB bus frequencies from the clock sources and prescalers selected by the user. Wrong settings are detected and highlighted in red through a dynamic validation of minimum and maximum conditions. Useful tooltips provide a detailed description of the actions to undertake when the settings are unavailable or wrong. User frequency selection can influence some peripheral parameters (e.g. UART baudrate limitation).

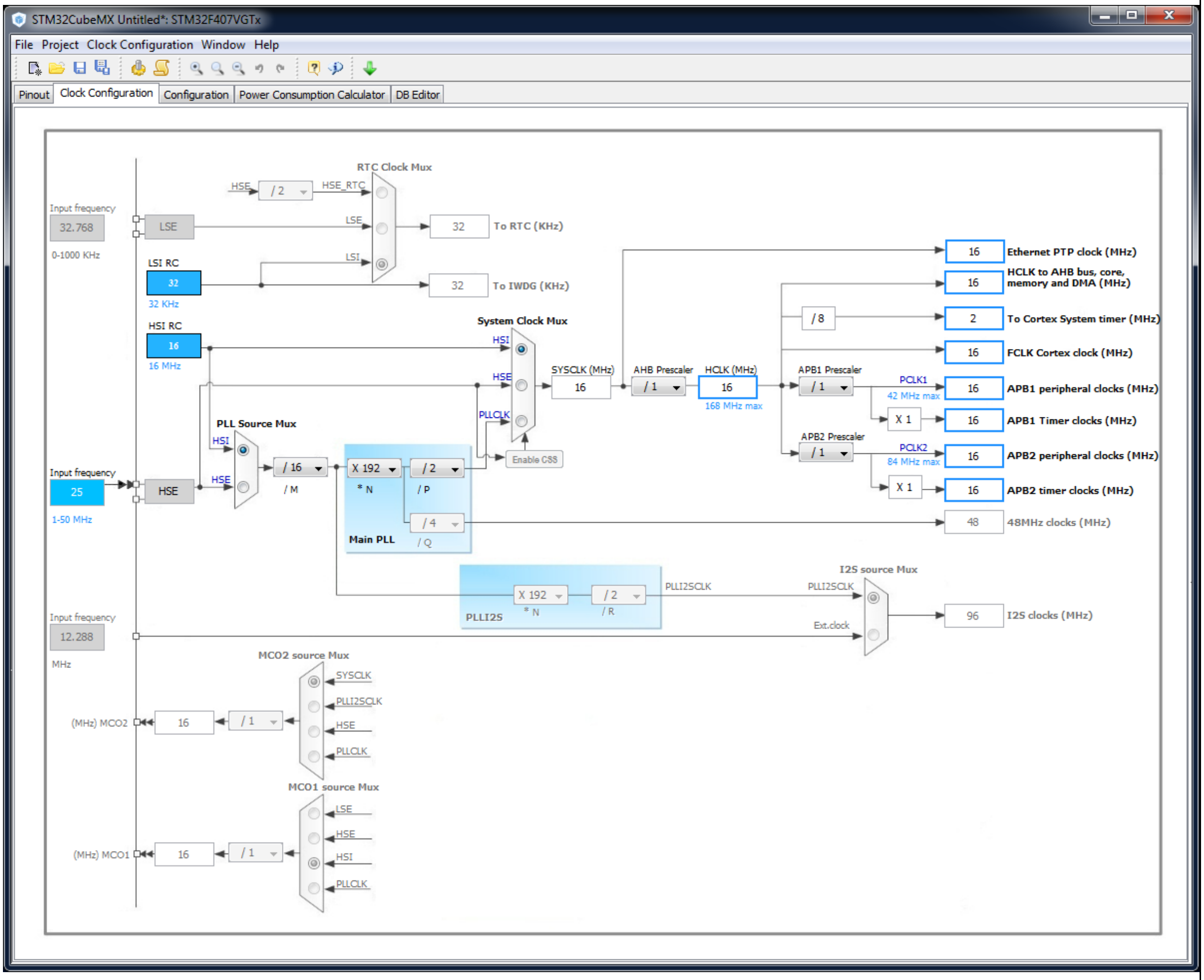
STM32CubeMX uses the clock settings defined in the Clock tree view to generate the initialization C code for each peripheral clock. Clock settings are performed in the generated C code as part of RCC initialization within the project main.c and in stm32f4xx_hal_conf.h (HSE, HSI and External clock values expressed in Hertz).

Follow the sequence below to configure the MCU clock tree:

1. Click the **Clock Configuration** tab to display the clock tree (see *Figure 126*).

   The internal (HSI, LSI), system (SYSCLK) and peripheral clock frequency fields cannot be edited. The system and peripheral clocks can be adjusted by selecting a clock source, and optionally by using the PLL, prescalers and multipliers.

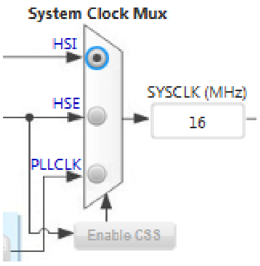
Figure 126. Clock tree view



2. First select the clock source (HSE, HSI or PLLCLK) that will drive the system clock of the microcontroller.

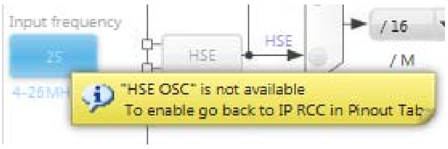
   In the example taken for the tutorial, select HSI to use the internal 16 MHz clock (see *Figure 127*).

Figure 127. HSI clock enabled



To use an external clock source (HSE or LSE), the RCC peripheral shall be configured in the **Pinout** view since pins will be used to connect the external clock crystals (see *Figure 128*).

Figure 128. HSE clock source disabled

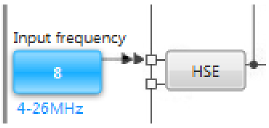


"HSE OSC" is not available
To enable go back to IP RCC in Pinout Tab

Other clock configuration options for the STM32F4DISCOVERY board would have been:

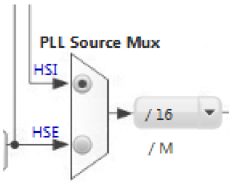
– To select the external HSE source and enter 8 in the HSE input frequency box since an 8 MHz crystal is connected on the discovery board:

Figure 129. HSE clock source enabled



– To select the external PLL clock source and the HSI or HSE as the PLL input clock source.

Figure 130. External PLL clock source enabled



3. Keep the core and peripheral clocks to 16 MHz using HSI, no PLL and no prescaling.

Note: *Optionally, further adjust the system and peripheral clocks using PLL, prescalers and multipliers:*

Other clock sources independent from the system clock can be configured as follows:

– *USB OTG FS, Random Number Generator and SDIO clocks are driven by an independent output of the PLL.*

– *I2S peripherals come with their own internal clock (PLLI2S), alternatively derived by an independent external clock source.*

– *USB OTG HS and Ethernet Clocks are derived from an external source.*

4. Optionally, configure the prescaler for the Microcontroller Clock Output (MCO) pins that allow to output two clocks to the external circuit.
5. Click 💾 to save the project.
6. Go to the **Configuration** tab to proceed with the project configuration.

6.6 Configuring the MCU initialization parameters

Reminder

The C code generated by STM32CubeMX covers the initialization of the MCU peripherals and middlewares using the STM32Cube firmware libraries.

6.6.1 Initial conditions

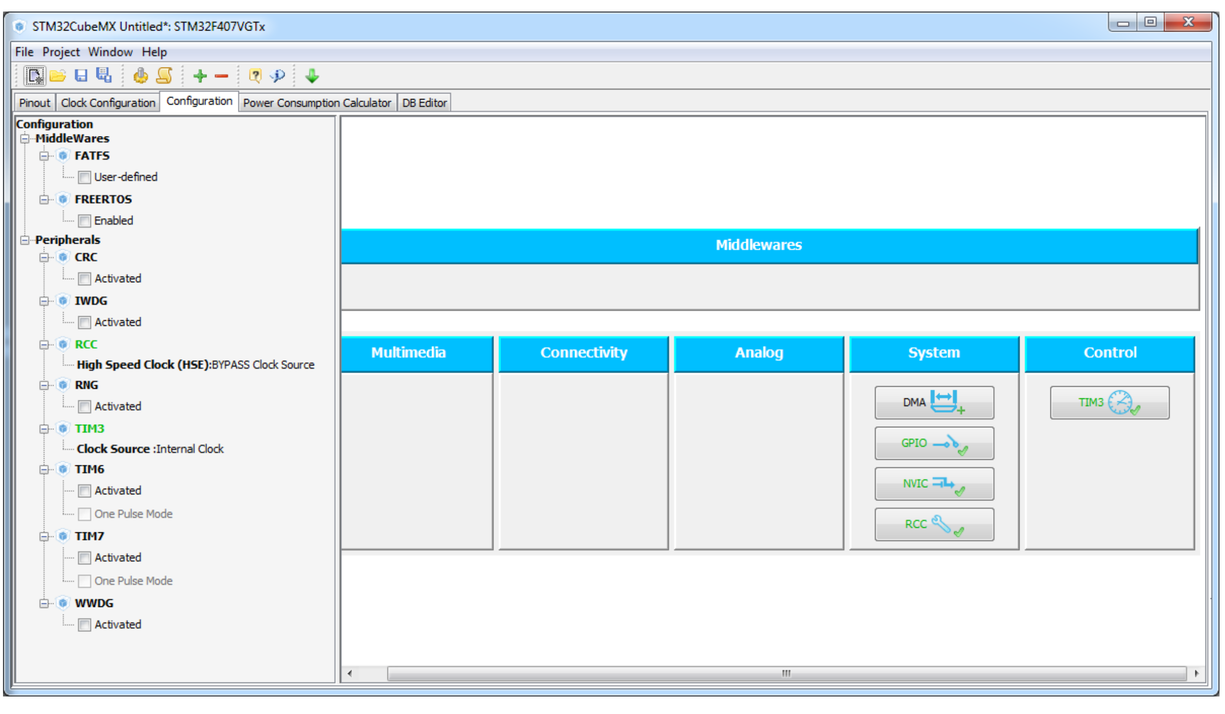
Select the **Configuration** tab to display the configuration view (see *Figure 131*).

Peripherals and middleware modes without influence on the pinout can be disabled or enabled in the **IP Tree** pane. The modes that impact the pin assignments can only be selected through the **Pinout** tab.

In the main panel, tooltips and warning messages are displayed when peripherals are not properly configured (see *Section 4: STM32CubeMX User Interface* for details).

Note: *The **RCC** peripheral initialization will use the parameter configuration done in this view as well as the configuration done in the Clock tree view (clock source, frequencies, prescaler values, etc…).*

Figure 131. Configuration view



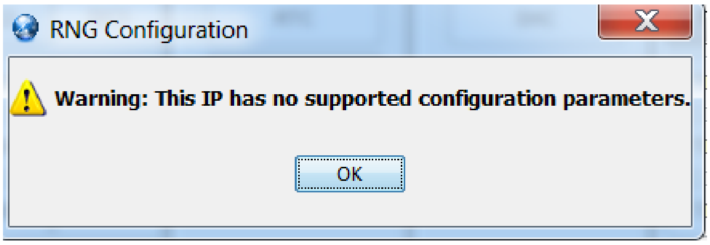
Configuration
- MiddleWares
  - FATFS
    - □ User-defined
  - FREERTOS
    - □ Enabled
- Peripherals
  - CRC
    - □ Activated
  - IWDG
    - □ Activated
  - RCC
    - High Speed Clock (HSE):BYPASS Clock Source
  - RNG
    - □ Activated
  - TIM3
    - Clock Source :Internal Clock
  - TIM6
    - □ Activated
    - □ One Pulse Mode
  - TIM7
    - □ Activated
    - □ One Pulse Mode
  - WWDG
    - □ Activated

6.6.2 Configuring the peripherals

Each peripheral instance corresponds to a dedicated button in the main panel.

Some peripheral modes have no configurable parameters as illustrated below:

Figure 132. Case of IP without configuration parameters



RNG Configuration

Warning: This IP has no supported configuration parameters.

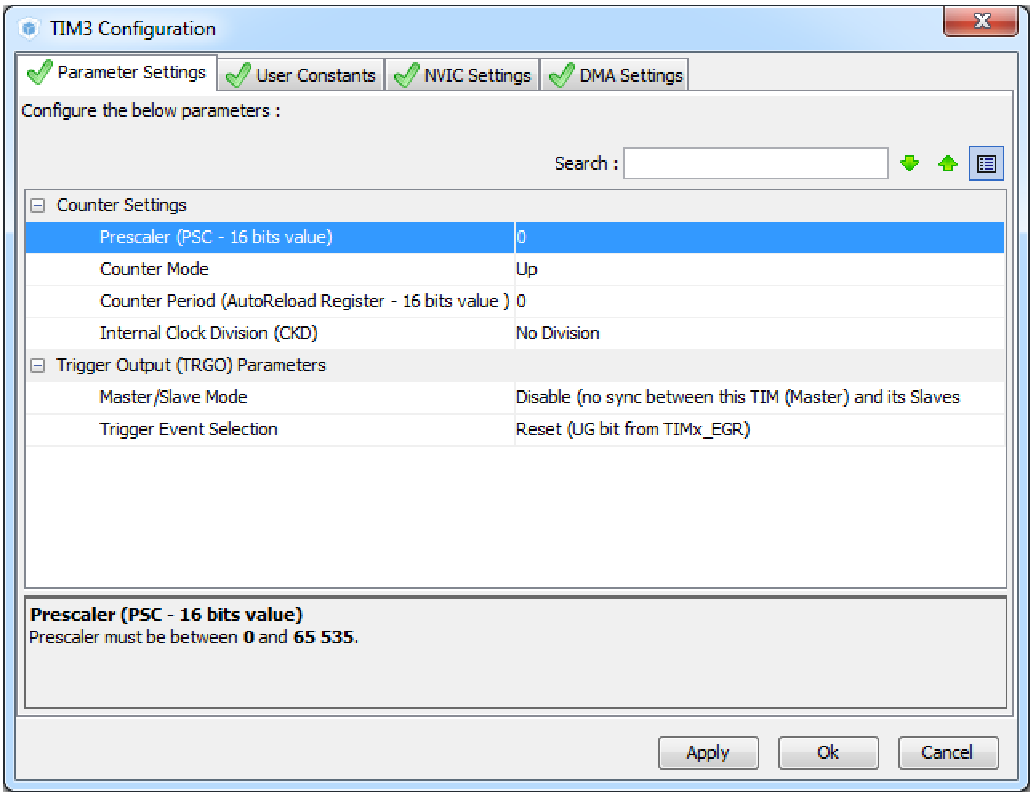
Follow the steps below to proceed with peripheral configuration:

1. Click the peripheral button to open the corresponding configuration window.

   In our example,

   a) Click TIM3 to open the timer configuration window.

Figure 133. Timer 3 configuration window



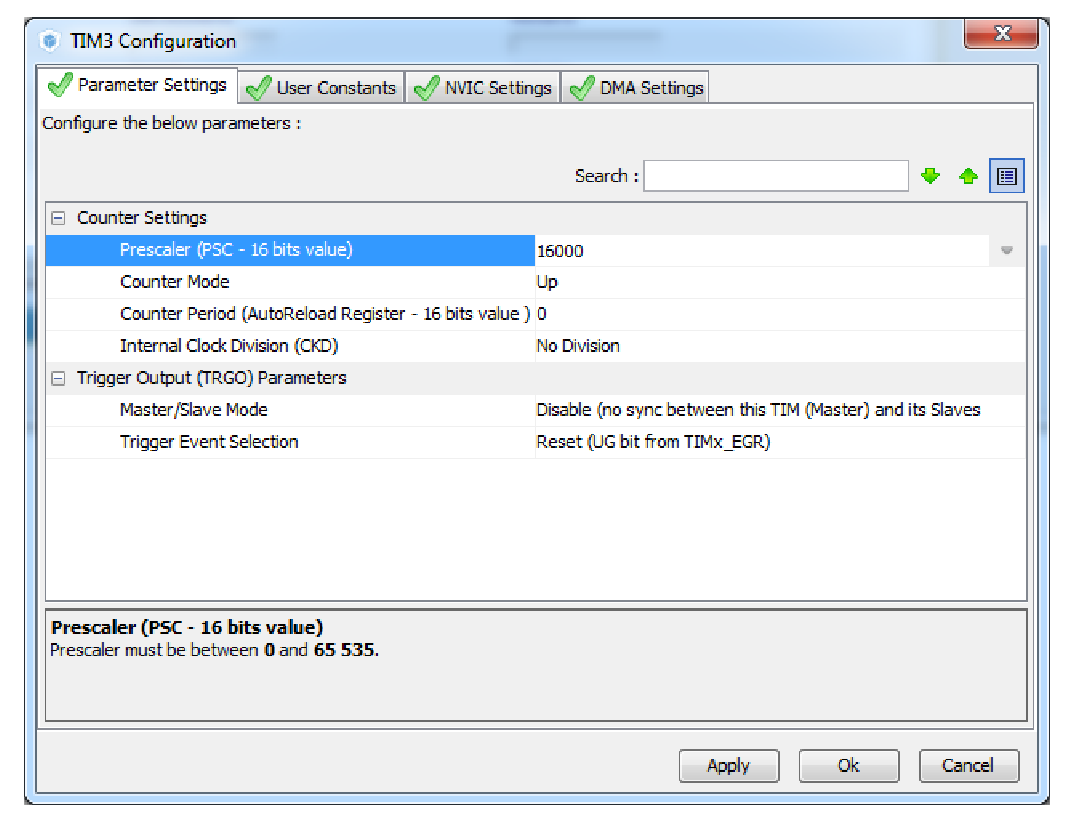
Configure the below parameters :

⊟ Counter Settings	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	0
Internal Clock Division (CKD)	No Division
⊟ Trigger Output (TRGO) Parameters	
Master/Slave Mode	Disable (no sync between this TIM (Master) and its Slaves
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

Prescaler (PSC - 16 bits value)
Prescaler must be between **0** and **65 535**.

b) With a 16 MHz APB clock (Clock tree view), set the prescaler to 16000 and the counter period to 1000 to make the LED blink every millisecond.

Figure 134. Timer 3 configuration



Configure the below parameters :

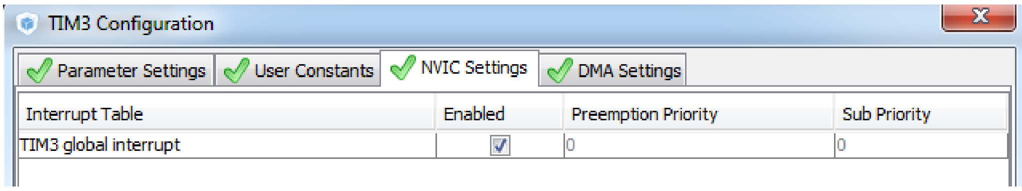
⊟ Counter Settings	
Prescaler (PSC - 16 bits value)	16000
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	0
Internal Clock Division (CKD)	No Division
⊟ Trigger Output (TRGO) Parameters	
Master/Slave Mode	Disable (no sync between this TIM (Master) and its Slaves
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

Prescaler (PSC - 16 bits value)
Prescaler must be between **0** and **65 535**.

2. Optionally and when available, select:
   - The **NVIC Settings** tab to display the NVIC configuration and enable interruptions for this peripheral.
   - The **DMA Settings** tab to display the DMA configuration and to configure DMA transfers for this peripheral.

     In the tutorial example, the DMA is not used and the GPIO settings remain unchanged. The interrupt is enabled as shown in *Figure 135*.
   - The **GPIO Settings** tab to display the GPIO configuration and to configure the GPIOs for this peripheral.
   - Insert an item:
   - The **User Constants** tab to specify constants to be used in the project.
3. Modify and click Apply or OK to save your modifications.

Figure 135. Enabling Timer 3 interrupt

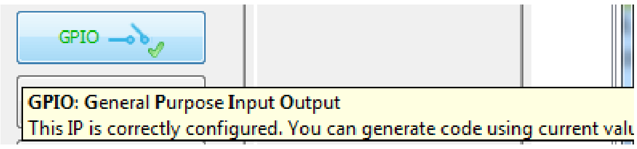


Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM3 global interrupt	☑	0	0

6.6.3 Configuring the GPIOs

The user can adjust all pin configurations from this window. A small icon along with a tooltip indicates the configuration status.

Figure 136. GPIO configuration color scheme and tooltip



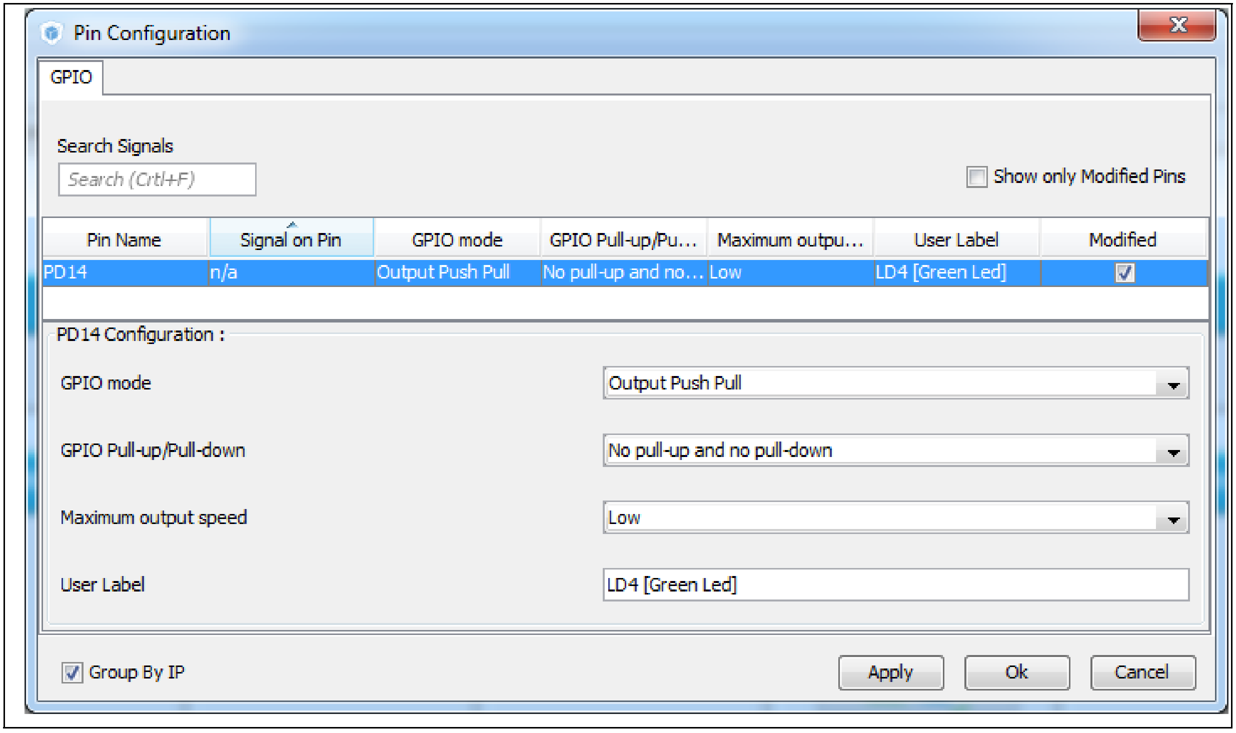
GPIO: General Purpose Input Output
This IP is correctly configured. You can generate code using current valu

Follow the sequence below to configure the GPIOS:

1. Click the **GPIO button** in the Configuration view to open the **Pin Configuration** window below.
2. The first tab shows the pins that have been assigned a GPIO mode but not for a dedicated IP. Select a Pin Name to open the configuration for that pin.

   In the tutorial example, select PD12 and configure it in output push-pull mode to drive the STM32F4DISCOVERY LED (see *Figure 137*).

Figure 137. GPIO mode configuration



Pin Name	Signal on Pin	GPIO mode	GPIO Pull-up/Pu...	Maximum outpu...	User Label	Modified
PD14	n/a	Output Push Pull	No pull-up and no...	Low	LD4 [Green Led]	☑

PD14 Configuration :

GPIO mode: Output Push Pull

GPIO Pull-up/Pull-down: No pull-up and no pull-down

Maximum output speed: Low

User Label: LD4 [Green Led]

3. Click **Apply** then **Ok** to close the window.

6.6.4 Configuring the DMAs

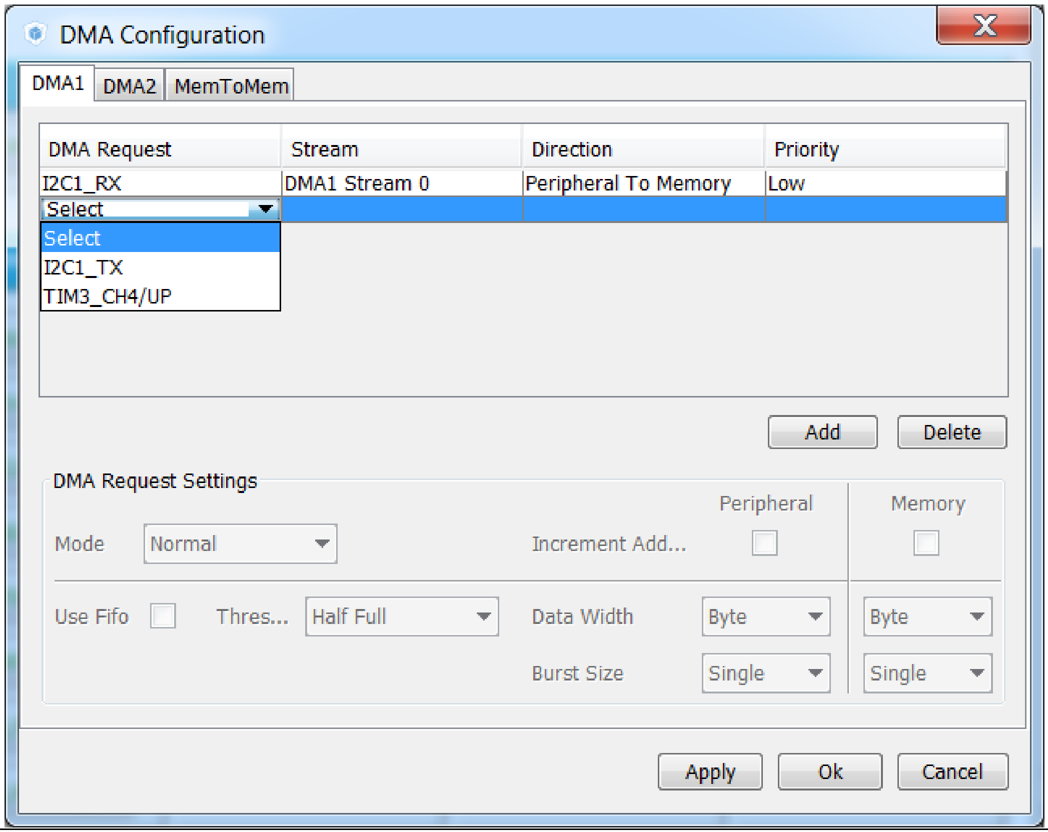
This is not required for the example taken for the tutorial.

It is recommended to use DMA transfers to offload the CPU. The DMA Configuration window provides a fast and easy way to configure the DMAs (see *Figure 138*).

1. Add a new DMA request and select among a list of possible configurations.
2. Select among the available streams.
3. Select the Direction: Memory to Peripheral or Peripheral to Memory.
4. Select a Priority.

Note: Configuring the DMA for a given IP can also be performed using the IP configuration window.

Figure 138. DMA Parameters configuration window



DMA Request	Stream	Direction	Priority
I2C1_RX	DMA1 Stream 0	Peripheral To Memory	Low

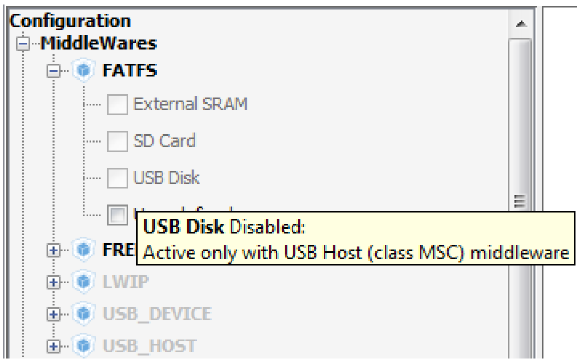
		Peripheral	Memory
Mode: Normal	Increment Add...	□	□
Use Fifo □ Thres... Half Full	Data Width	Byte	Byte
	Burst Size	Single	Single

6.6.5 Configuring the middleware

This is not required for the example taken for the tutorial.

If a peripheral is required for a middleware mode, the peripheral must be configured in the **Pinout** view for the middleware mode to become available. A tooltip can guide the user as illustrated in the FatFs example below:

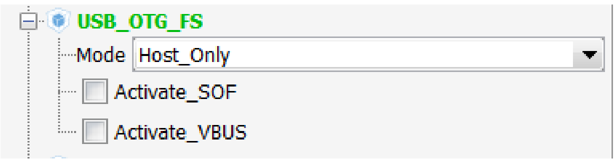
Figure 139. FatFs disabled



USB Disk Disabled:
Active only with USB Host (class MSC) middleware

1. Configure the USB IP from the **Pinout** view.

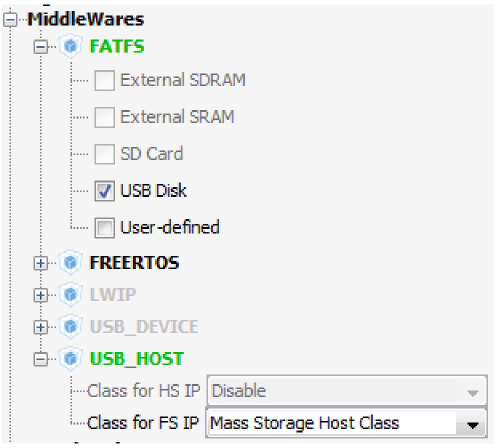
Figure 140. USB Host configuration



USB_OTG_FS
- Mode: Host_Only
- □ Activate_SOF
- □ Activate_VBUS

2. Select MSC_FS class from USB Host middleware.
3. Select the checkbox to enable FatFs USB mode in the tree panel.

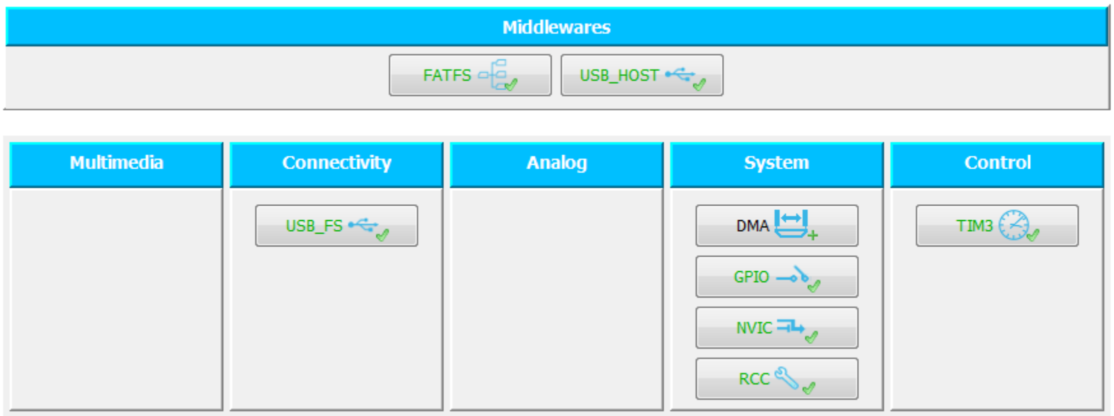
Figure 141. FatFs over USB mode enabled



MiddleWares
- FATFS
  - □ External SDRAM
  - □ External SRAM
  - □ SD Card
  - ☑ USB Disk
  - □ User-defined
- FREERTOS
- LWIP
- USB_DEVICE
- USB_HOST
  - Class for HS IP: Disable
  - Class for FS IP: Mass Storage Host Class

4. Select the **Configuration** view. FatFs and USB buttons are then displayed.

Figure 142. Configuration view with FatFs and USB enabled


Middlewares
FATFS
USB_HOST
Multimedia
Connectivity
USB_FS
Analog
System
DMA
GPIO
NVIC
RCC
Control
TIM3