# UM1718
# User manual

## STM32CubeMX for STM32 configuration and initialization C code generation

## Introduction

STM32CubeMX is a graphical tool for 32-bit ARM® Cortex® STM32 microcontrollers. It is part of STMCube™ initiative (see *Section 1*) and is available either as a standalone application or as an Eclipse plug-in for integration in Integrated Development Environments (IDEs).

STM32CubeMX has the following key features:

- **Easy microcontroller selection covering whole STM32 portfolio.**
- **Board selection from a list of STMicroelectronics boards.**
- **Easy microcontroller configuration** (pins, clock tree, peripherals, middleware) and generation of the corresponding initialization C code.
- **Generation of configuration reports**.
- **Generation of IDE ready projects** for a selection of integrated development environment tool chains.

  STM32CubeMX projects include the generated initialization C code, STM32 HAL drivers, the middleware stacks required for the user configuration, and all the relevant files needed to open and build the project in the selected IDE.
- **Power consumption calculation** for a user-defined application sequence**.**
- **Self-updates** allowing the user to keep the STM32CubeMX up-to-date.
- Download and update of STM32Cube™ embedded software required for user application development (see *Appendix E: STM32Cube embedded software packages* for details on STM32Cube embedded software offer).

Although STM32CubeMX offers a user interface and generates a C code compliant with STM32 MCU design and firmware solutions, it is recommended to refer to the product technical documentation for details on actual implementation of microcontroller peripherals and firmware.

### Reference documents

The following documents are available from *http://www.st.com*:

- STM32 microcontroller reference manuals
- STM32 microcontroller datasheets
- *STM32Cube HAL driver user manuals for STM32F0xx (UM1785), STM32F1xx (UM1850), STM32F2xx (UM1742), STM32F3xx (UM1786), STM32F4xx (UM1725), STM32L0xx (UM1749) and STM32L1xx (UM1816).*

# Contents

# List of tables

# List of figures

# 1 STM32Cube overview

STMCube™ is an STMicroelectronics original initiative to ease developers life by reducing development efforts, time and cost. STM32Cube covers STM32 portfolio.

STM32Cube includes:

- The STM32CubeMX, a graphical software configuration tool that allows to generate C initialization C code using graphical wizards.
- A comprehensive embedded software platform, delivered per series (such as STM32CubeF2 for STM32F2 series and STM32CubeF4 for STM32F4 series)
  - The STM32Cube HAL, an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, Graphics
  - All embedded software utilities coming with a full set of examples.

# 2 Getting started with STM32CubeMX

## 2.1 Principles

Customers need to quickly identify the MCU that best meets their requirements (core architecture, features, memory size, performance…). While board designers main concerns are to optimize the microcontroller pin configuration for their board layout and to fulfill the application requirements (choice of peripherals operating modes), embedded system developers are more interested in developing new applications for a specific target device, and migrating existing designs to different microcontrollers.

The time taken to migrate to new platforms and update the C code to new firmware drivers adds unnecessary delays to the project. STM32CubeMX was developed within STM32Cube initiative which purpose is to meet customer key requirements to maximize software reuse and minimize the time to create the target system:

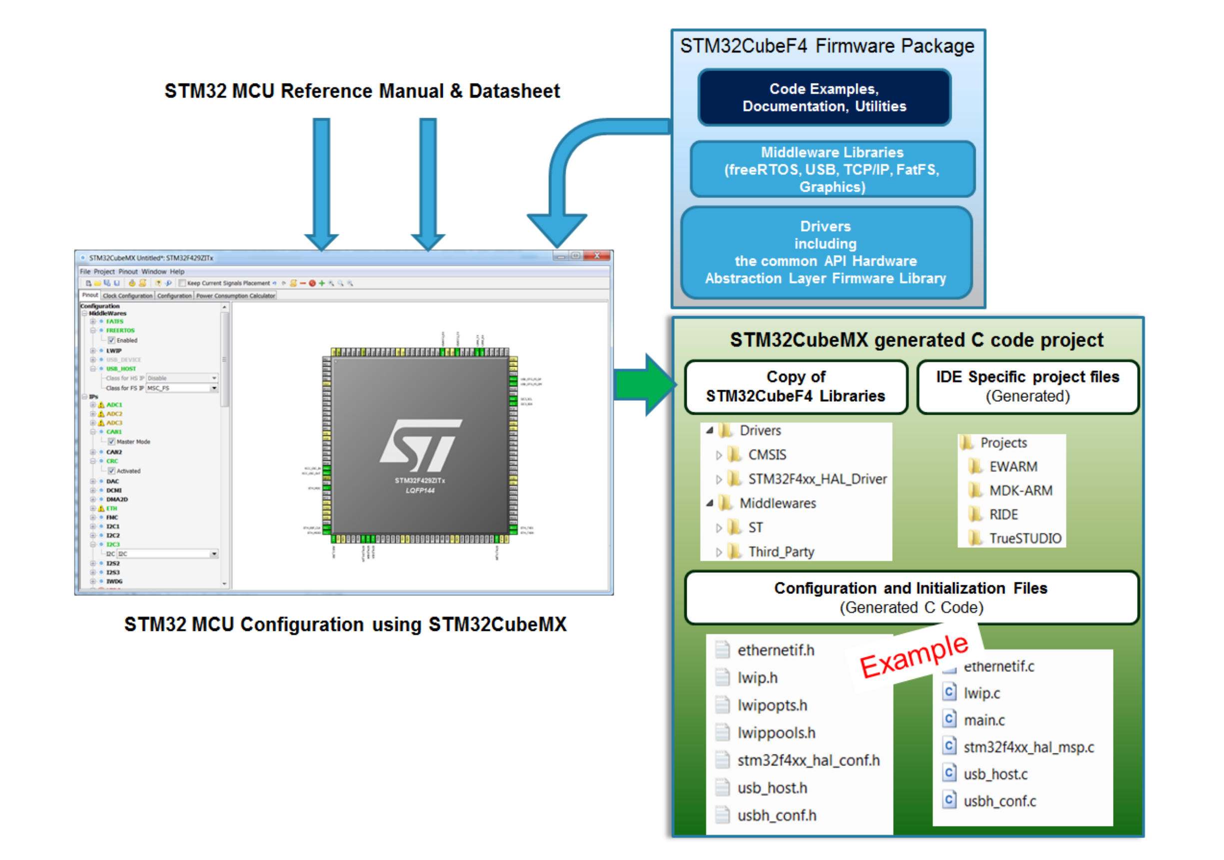
- Software reuse and application design portability are achieved through STM32Cube firmware solution proposing a common Hardware Abstraction Layer API across STM32 portfolio.
- Optimized migration time is achieved thanks to STM32CubeMX built-in knowledge of STM32 microcontrollers, peripherals and middleware (LwIP and USB communication protocol stacks, FATFS file system for small embedded systems, FreeRTOS).

STM32CubeMX graphical interface performs the following functions:

- Fast and easy configuration of the MCU pins, clock tree and operating modes for the selected peripherals and middleware
- Generation of pin configuration report for board designers
- Generation of a complete project with all the necessary libraries and initialization C code to set up the device in the user defined operating mode. The project can be directly imported in the selected application development environment (for a selection of supported IDEs) to proceed with application development (see *Figure 1*).

During the configuration process, STM32CubeMX detects conflicts and invalid settings and highlights them through meaningful icons and useful tool tips.

**Figure 1. Overview of STM32CubeMX C code generation flow**



## 2.2 Key features

STM32CubeMX comes with the following features:

- **Project management**

  STM32CubeMX allows creating, saving and loading previously saved projects:
  – When STM32CubeMX is launched, the user can choose to create a new project or to load a previously saved project.
  – Saving the project saves user settings and configuration performed within the project in an .ioc file that will be used the next time the project will be loaded in STM32CubeMX.

  STM32CubeMX projects come in two flavors:
  – MCU configuration only: .ioc file can be saved anywhere
  – MCU configuration with C code generation: in this case .ioc files are saved in a project dedicated folder along with the generated source C code.

- **Easy MCU and STMicroelectronics board selection**

  When starting a new project, a dedicated window opens to select either a microcontroller or an STMicroelectronics board from STM32 portfolio. Different filtering options are available to ease the MCU and board selection.

- **Easy pinout configuration**
  – From the **Pinout** view, the user can select the peripherals from a list and configure the peripheral modes required for the application. STM32CubeMX assigns and configures the pins accordingly.
  – For more advanced users, it is also possible to directly map a peripheral function to a physical pin using the **Chip** view. The signals can be locked on pins to prevent STM32CubeMX conflict solver from moving the signal to another pin.
  – Pinout configuration can be exported as a .csv file.

- **Complete project generation**

  The project generation includes pinout, firmware and middleware initialization C code for a set of IDEs. It is based on STM32Cube embedded software libraries. The following actions can be performed:
  – Starting from the previously defined pinout, the user can proceed with the configuration of middleware, clock tree, services (RNG, CRC, etc...) and IP peripheral parameters. STM32CubeMX generates the corresponding initialization C code. The result is a project directory including generated main.c file and C header files for configuration and initialization, plus a copy of the necessary HAL and middleware libraries as well as specific files for the selected IDE.
  – The user can modify the generated source files by adding user-defined C code in user dedicated sections. STM32CubeMX ensures that the user C code is preserved upon next C code generation (the user C code is commented if it is no longer relevant for the current configuration).
  – From the Project settings menu, the user can select the development tool chain (IDE) for which the C code has to be generated. STM32CubeMX ensures that the IDE relevant project files are added to the project folder so that the project can be

directly imported as a new project within third party IDE (IAR™ EWARM, Keil™ MDK-ARM, Atollic® TrueStudio and AC6 System Workbench for STM32).

- **Power consumption calculation**

  Starting with the selection of a microcontroller part number and a battery type, the user can define a sequence of steps representing the application life cycle and parameters (choice of frequencies, enabled peripherals, step duration). STM32CubeMX power consumption calculator returns the corresponding power consumption and battery life estimates.
- **Clock tree configuration**

  STM32CubeMX offers a graphical representation of the clock tree as it can be found in the device reference manual. The user can change the default settings (clock sources, prescaler and frequency values). The clock tree will be updated accordingly. Invalid settings and limitations are highlighted and documented with tool tips.
- **Automatic updates of STM32CubeMX and STM32Cube firmware packages**

  STM32CubeMX comes with an updater mechanism that can be configured for automatic or on-demand check for updates. It supports STM32CubeMX self-updates as well as STM32Cube firmware library package updates.
- **Report generation**

  .pdf and .csv reports can be generated to document user configuration work.

## 2.3 Rules and limitations

- C code generation covers only peripheral and middleware initialization. It is based on STM32Cube HAL firmware libraries.
- STM32CubeMX configuration files (.ioc extension) can be saved in the same folder when they target pin configuration only but require a dedicated folder when they cover configuration for C code generation.
- Refer to *Appendix A* for a description of pin assignment rules.
- Refer to *Appendix B* for a description of STM32CubeMX C code generation design choices and limitations.

# 3 Installing and running STM32CubeMX

## 3.1 System requirements

### 3.1.1 Supported operating systems and architectures

- Windows® XP: 32-bit (x86)
- Windows® 7: 32-bit (x86), 64-bit (x64)
- Windows® 8: 32-bit (x86), 64-bit (x64)

### 3.1.2 Memory prerequisites

- Recommended minimum RAM: 2 Gbytes.

### 3.1.3 Software requirements

The following software must be installed:

- Java Run Time Environment 1.7 (version 1.7_45 or newer)
  If Java is not installed on your computer or if you have an old version, STM32CubeMX installer will open the Java download web page and stop.
- For Eclipse plug-in installation only, install one of the following IDE:
  – Eclipse IDE Juno (4.2)
  – Eclipse Luna (4.4)
  – Eclipse Kepler (4.3)

## 3.2 Installing/uninstalling STM32CubeMX standalone version

### 3.2.1 Installing STM32CubeMX standalone version

To install STM32CubeMX, follow the steps below:

1. Download the latest STM32CubeMX installation package from http://www.st.com/stm32cube.
2. Download *STM32CubeMX-setup.zip* to your local disk and extract the *STM32CubeMX-setup.exe* file.
3. Double-click *STM32CubeMX-setup.exe* to launch the installation wizard.
4. If the proper version of the Java Runtime Environment (version 1.7_45 or newer) is not installed, the wizard will propose to download it and stop. Restart STM32CubeMX installation once Java installation is complete. Refer to *Section 9: FAQ* for issues when installing the JRE.
5. If the installation was successful, the STM32CubeMX icon is displayed on the desktop and STM32CubeMX application is available from the Program menu. STM32CubeMX .ioc files are displayed with a cube icon and double-clicking them opens up them using STM32CubeMX.

*Note: Only the latest installation of STM32CubeMX will be enabled in the program menu. Previous versions can be kept on your PC (not recommended) when different installation folders have been selected. Otherwise, the new installation overwrites the previous ones.*

### 3.2.2 Uninstalling STM32CubeMX standalone version

To uninstall STM32CubeMX, follow the steps below:

1. Open the Windows Control panel**.**
2. Select Programs and Features to display the list of programs installed on your computer.
3. Right click on STM32CubeMX and select the uninstall function.

## 3.3 Installing STM32CubeMX plug-in version

STM32CubeMX plug-in can be installed within Eclipse IDE development tool chain. Installation related procedures are described in this section.

### 3.3.1 Downloading STM32CubeMX plug-in installation package

To download STM32CubeMX plug-in, follow the sequence below:

1. Go to http://www.st.com/stm32cube.
2. Download STM32CubeMX- Eclipse-plug-in .zip file to your local disk.

### 3.3.2 Installing STM32CubeMX as an Eclipse IDE plug-in

To install STM32CubeMX as an Eclipse IDE plug-in, follow the sequence below:

1. Launch the Eclipse environment.
2. Select Help > Install New Software from the main menu bar. The Available Software window appears.
3. Click Add. The Add Repository window opens.
4. Click Archive. The Repository archive browser opens.
5. Select the STM32CubeMX- Eclipse-plug-in .zip file that you downloaded and click Open (see *Figure 2*).
6. Click OK in the Add Repository dialog box,
7. Check STM32CubeMX_Eclipse_plug-in and click Next (see *Figure 3*).
8. Click Next in the Install Details dialog box.
9. Click ”I accept the terms of the license agreement” in the Review Licenses dialog box and then click Finish.
10. Click OK in the Security Warning menu.
11. Click OK when requested to restart Eclipse IDE (see *Section 3.4.2: Running STM32CubeMX plug-in from Eclipse IDE*).

**Figure 2. Adding STM32CubeMX plug-in archive**

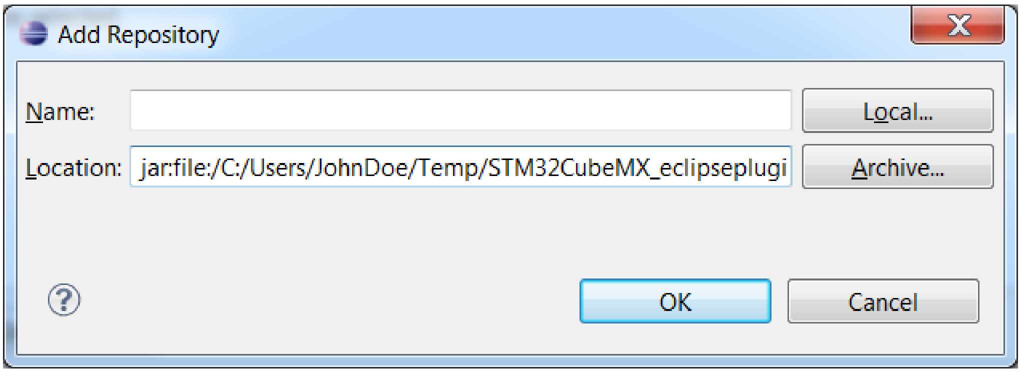
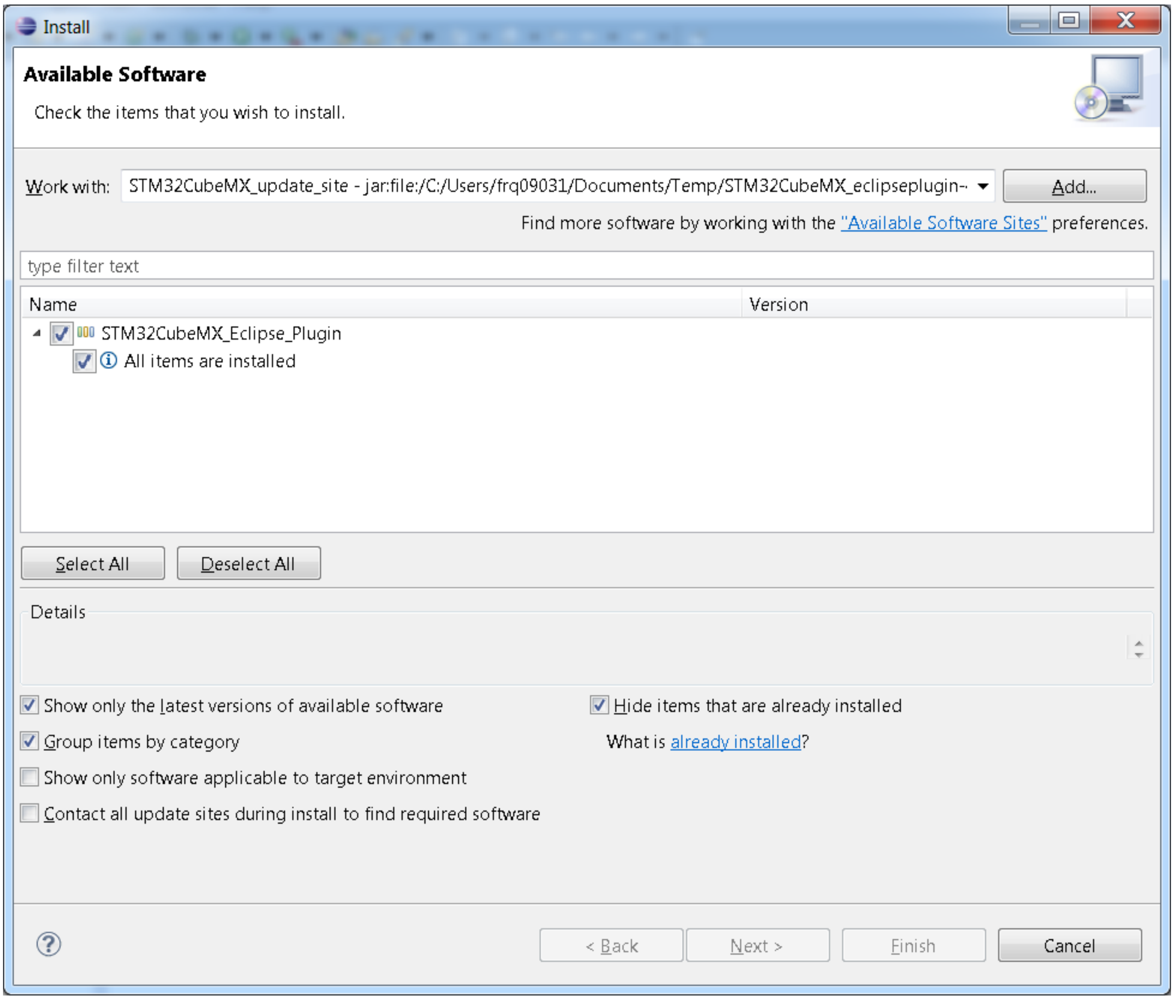



**Figure 3. Installing STM32CubeMX plug-in**



### 3.3.3 Uninstalling STM32CubeMX as an Eclipse IDE plug-in

To uninstall STM32CubeMX plug-in in Eclipse IDE, follow the sequence below:

1. In Eclipse, right-click STM32CubeMX perspective Icon (see *Figure 4*) and select Close.
2. From Eclipse Help menu, select Install New Software.
3. Click the Installed Software tab, then select STM32CubeMX and click Uninstall.
4. Click Finish in the Uninstall Details menu (see *Figure 5*).

**Figure 4. Closing STM32CubeMX perspective**

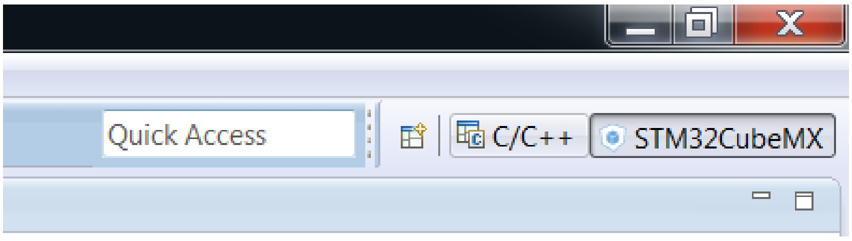
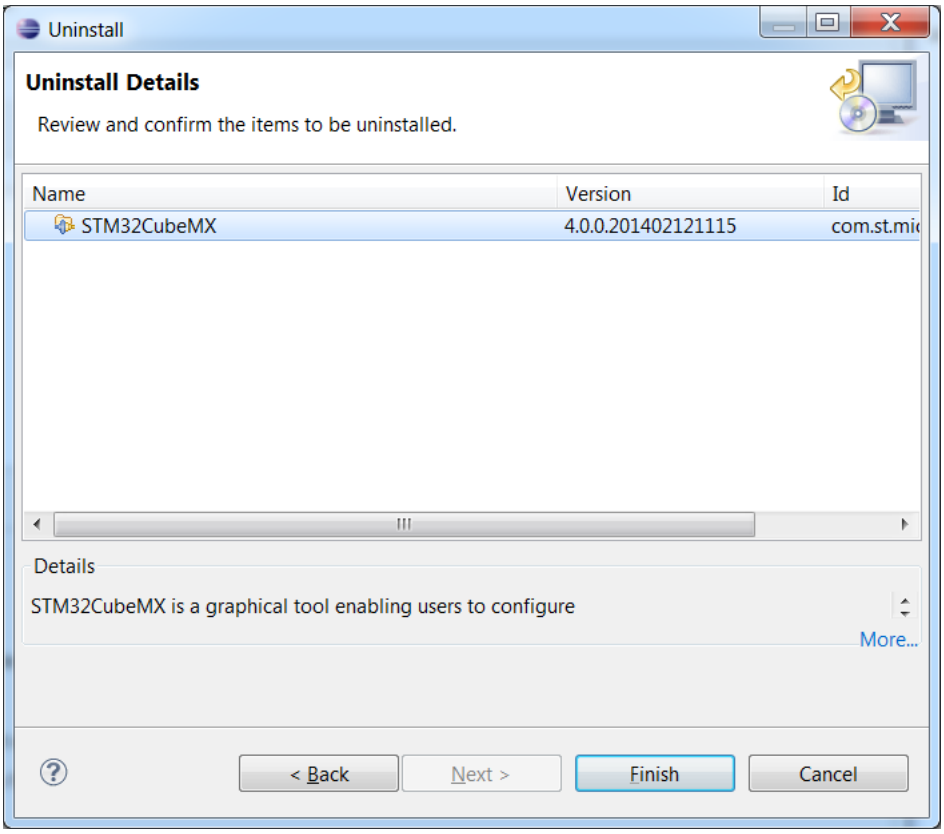



**Figure 5. Uninstalling STM32CubeMX plug-in**



## 3.4 Launching STM32CubeMX

### 3.4.1 Running STM32CubeMX as standalone application

To run STM32CubeMX as a standalone application:

- Select STM32CubeMX from Program Files > ST Microelectronics > STM32CubeMX.
- Or double-click STM32CubeMX icon on your desktop.

### 3.4.2 Running STM32CubeMX plug-in from Eclipse IDE

To run STM32CubeMX plug-in from Eclipse:

1. Launch Eclipse environment.
2. Once Eclipse IDE is open, click open new perspective: .
3. Select STM32CubeMX to open STM32CubeMX as a perspective (see *Figure 6*).
4. STM32CubeMX perspective opens (see *Figure 7*). Enter STM32CubeMX user interface via the Welcome menus.

**Figure 6. Opening Eclipse plug-in**

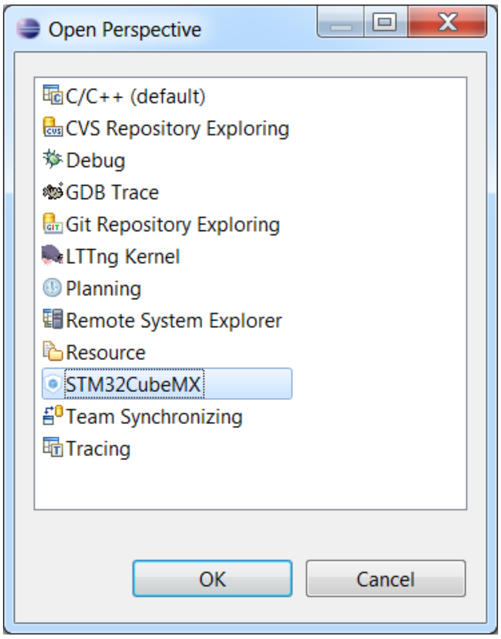
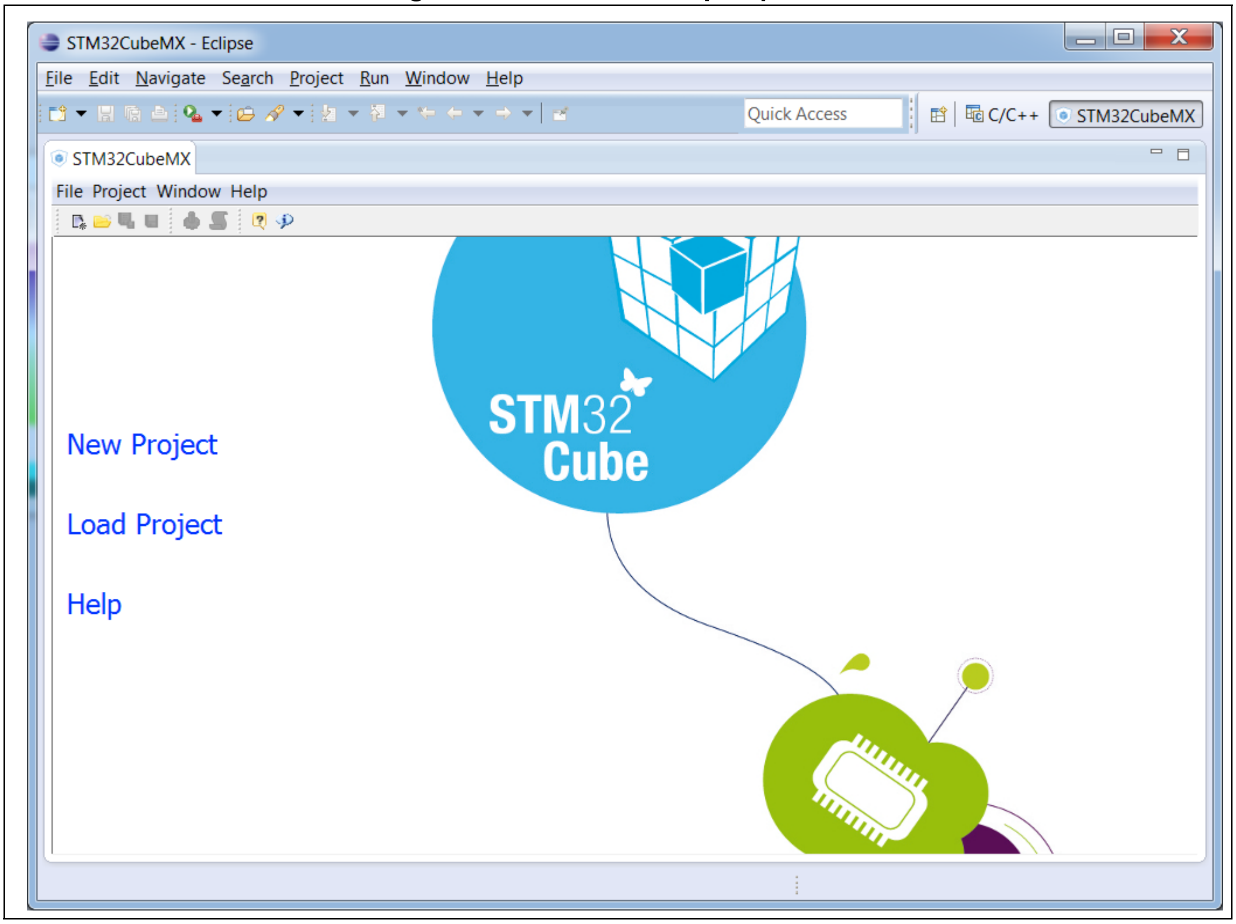


Figure 7. STM32CubeMX perspective




## 3.5 Getting STM32Cube updates

STM32CubeMX implements a mechanism to access the internet and to:

- Perform self-updates of STM32CubeMX and of the STM32Cube firmware packages installed on the user computer
- Download new firmware packages and patches

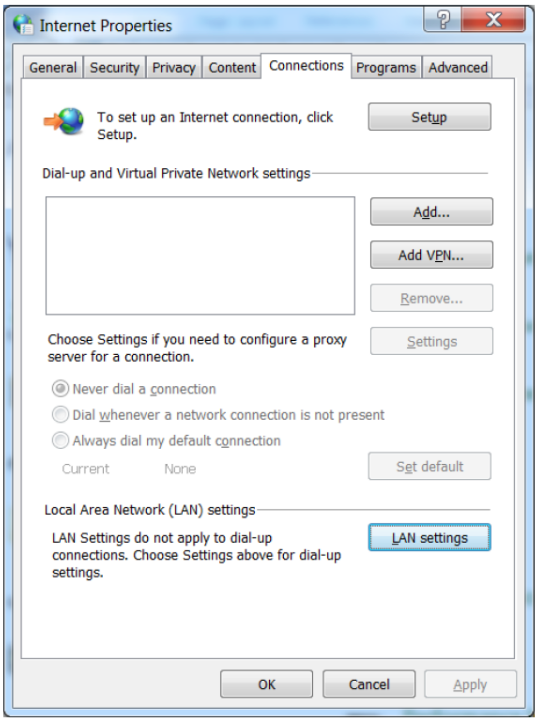
Installation and update related sub-menus are available under the Help menu.

Off-line updates can also be performed on computers without internet access (see *Figure 13*). This is done by browsing the filesystem and selecting available STM32Cube firmware zip packages.

If the PC on which STM32CubeMX runs is connected to a computer network using a proxy server, STM32CubeMX needs to connect to that server to access the internet, get self-updates and download firmware packages. Refer to *Section 3.5.1: Updater configuration* for a description of this connection configuration.

To view Windows default proxy settings, select Internet options from the Control panel and select LAN settings from the Connections tab (see *Figure 8*).

**Figure 8. Displaying Windows default proxy settings**




Several proxy types exist and different computer network configurations are possible:

- Without proxy: the application directly accesses the web (Windows default configuration).
- Proxy without login/password
- Proxy with login/password: when using an internet browser, a dialog box opens and prompts the user to enter his login/password.
- Web proxies with login/password: when using an internet browser, a web page opens and prompts the user to enter his login/password.

If necessary, contact your IT administrator for proxy information (proxy type, http address, port).

STM32CubeMX does not support web proxies. In this case, the user will not be able to benefit from the update mechanism and will need to manually copy the STM32 firmware packages from http://www.st.com/stm32cube to the repository. To do it, follow the sequence below:

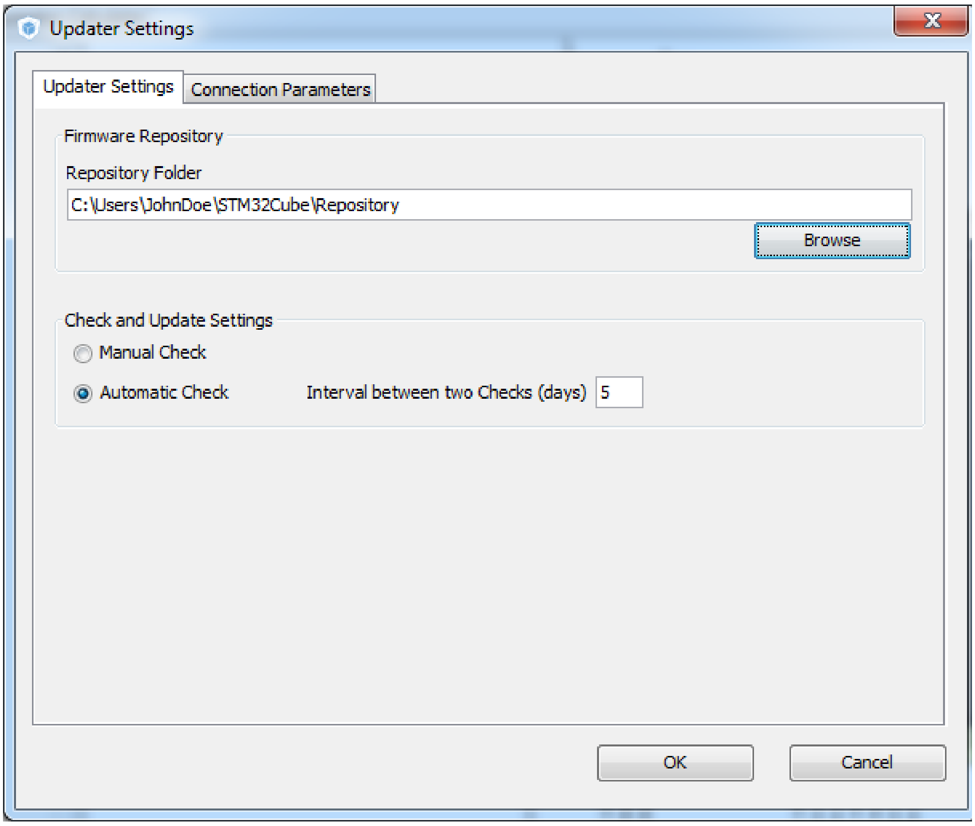
1. Go to http://www.st.com/stm32cube and download the relevant STM32Cube firmware package from the *Associated Software* section.
2. Unzip the zip package to your STM32Cube repository. Find out the default repository folder location in the Updater settings tab as shown in *Figure 9* (you might need to update it to use a different location or name).

### 3.5.1 Updater configuration

To perform STM32Cube new library package installation or updates, the tool must be configured as follows:

1. Select **Help > Updater Settings** to open the **Updater Settings** window.
2. From the **Updater Settings** tab (see *Figure 9*)
   a) Specify the repository destination folder where the downloaded packages will be stored.
   b) Enable/Disable the automatic check for updates.

**Figure 9. Updater Settings tab**




3. In the **Connection Parameters** tab, specify the proxy server settings appropriate for your network configuration by selecting a proxy type among the following possibilities:
   – No Proxy (see *Figure 10*)
   – Use System Proxy Parameters (see *Figure 11*)
     On Windows, proxy parameters will be retrieved from the PC system settings.
     Uncheck “Require Authentication” if a proxy server without login/password configuration is used.
   – Manual Configuration of Proxy Server (see *Figure 12*)
     Enter the Proxy server http address and port number. Enter login/password information or uncheck “Require Authentication” if a proxy server without login/password configuration is used.
4. Click the Check Connection button to verify if the connection works. A green check mark appears to confirm that the connection operates correctly 

:

**Figure 10. Connection Parameters tab - No proxy**

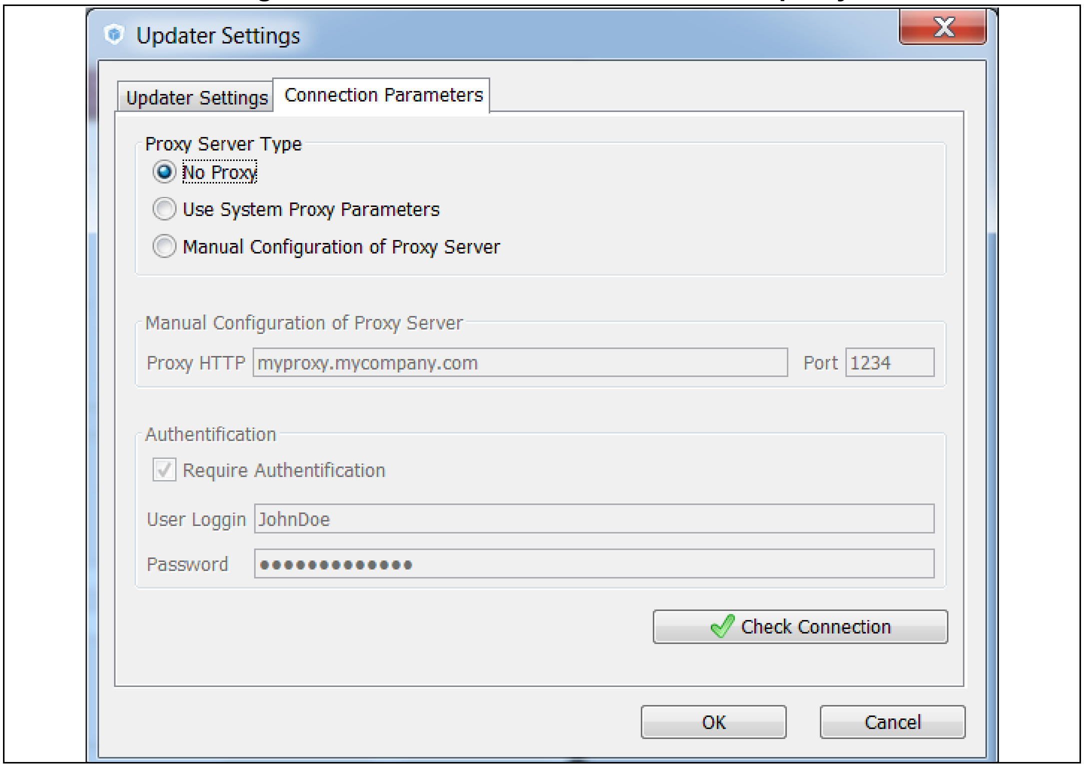



**Figure 11. Connection Parameters tab - Use System proxy parameters**

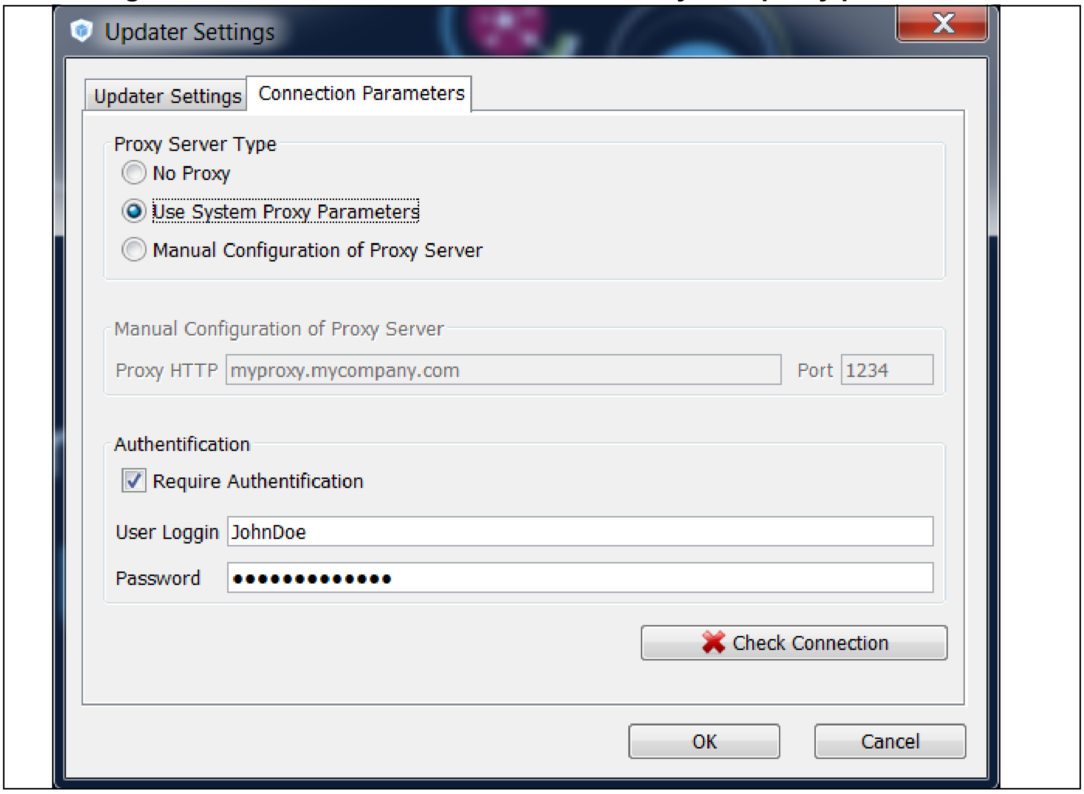
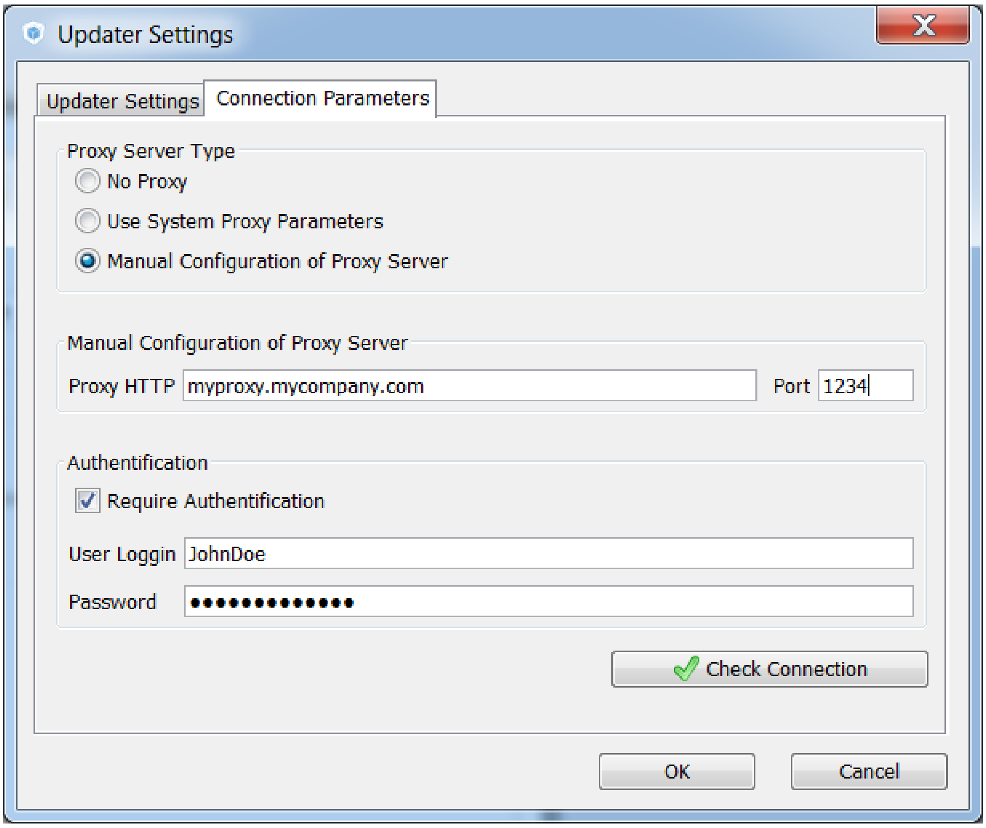


**Figure 12. Connection Parameters tab - Manual Configuration of Proxy Server**




5. Select **Help > Install New Libraries** sub-menu to select among a list of possible packages to install.
6. If the tool is configured for manual checks, select **Help > Check for Updates** to find out about new tool versions or firmware library patches available to install.

### 3.5.2 Downloading new libraries

To download new libraries, follow the steps below:

1. Select **Help > Install New Libraries** to open the New Libraries Manager window.

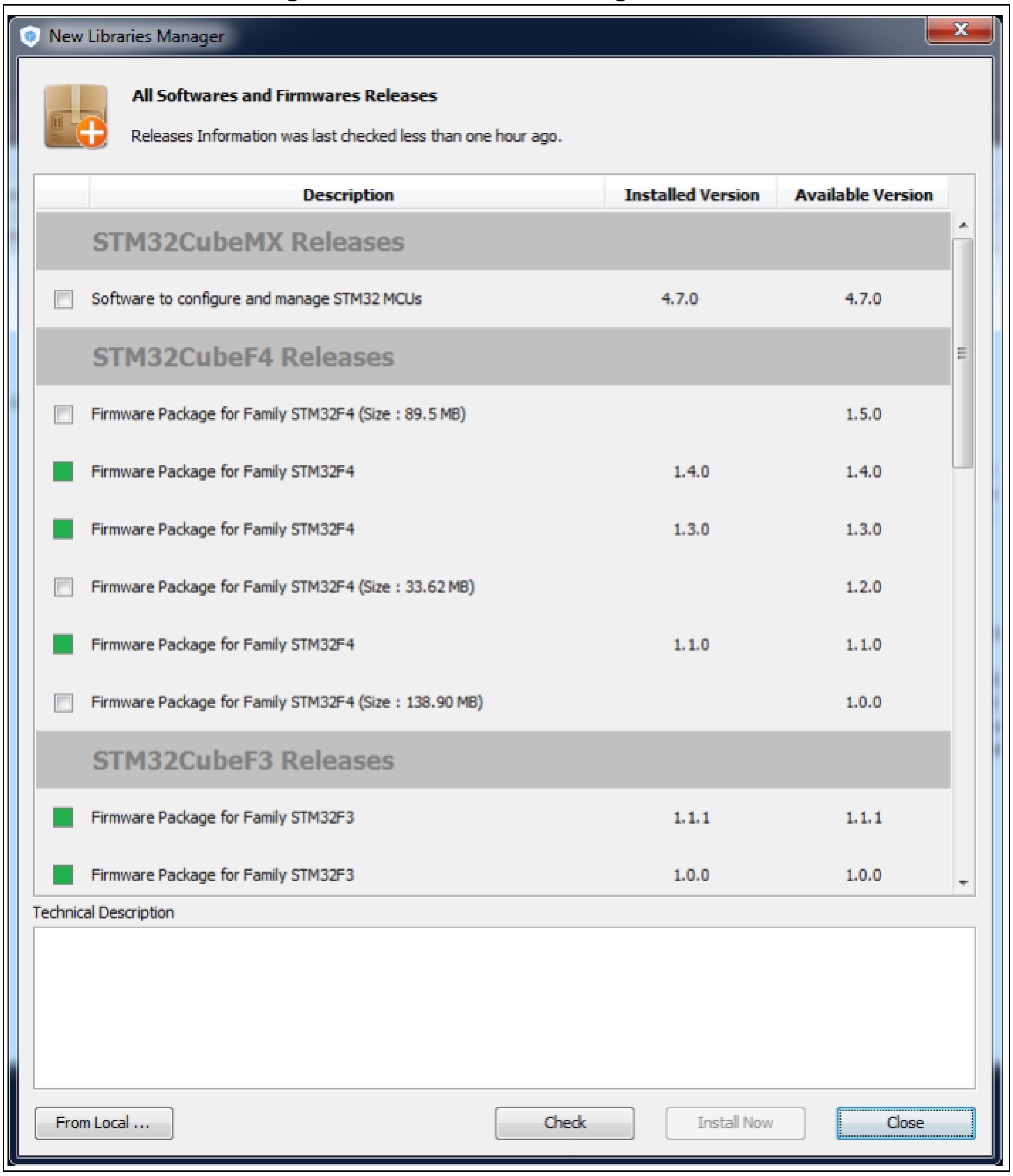
   If the installation was performed using STM32CubeMX, all the packages available for download are displayed along with their version including the version currently installed on the user PC (if any), and the latest version available from *http://www.st.com*.

   If no Internet access is available at that time, choose “Local File”. Then, browse to select the zip file of the desired STM32Cube firmware package that has been previously downloaded from st.com. An integrity check is performed on the file to ensure that it is fully supported by STM32CubeMX.

   The package is marked in green when the version installed matches the latest version available from *http://www.st.com*.
2. Click the checkbox to select a package then “Install Now” to start the download.

See *Figure 13* for an example.

**Figure 13. New libraries Manager window**




### 3.5.3 Checking for updates

When the updater is configured for automatic checks, it regularly verifies if updates are available. In this case, a green arrow icon appears on the tool bar.

When automatic checks have been disabled in the updater settings window, the user can manually check if updates are available:

1. Click the icon to open the **Update Manager** window or Select **Help > Check for Updates**. All the updates available for the user current installation are listed.
2. Click the check box to select a package, and then Install Now to download the update.

# 4 STM32CubeMX User Interface

STM32CubeMX user interface consists of a main window, a menu bar, a toolbar, four views (Pinout, Configuration, Clock Configuration, Power Consumption Calculator) and a set of help windows (MCUs selection, Update manager, About). All these menus are described in the following sections.

For C code generation, although the user can switch back and forth between the different configuration views, it is recommended to follow the sequence below:

1. Select the relevant IPs and their operating modes from the **Pinout** view.
2. Configure the clock tree from the clock configuration view.

   In the **Pinout** view, configure the RCC peripheral by enabling the external clocks, master output clocks, audio input clocks (when relevant for your application). This automatically displays more options on the **Clock tree** view (see *Figure 17*).
3. Configure the parameters required to initialize the IP operating modes from the configuration view.
4. Generate the initialization C code.

## 4.1 Welcome page

The Welcome page is the first window that opens up when launching STM32CubeMX program. It remains open as long as the application is running. Closing it closes down the application. Refer to *Figure 14* and to *Table 1* for a description of the Welcome page.

**Figure 14. STM32CubeMX Welcome page**

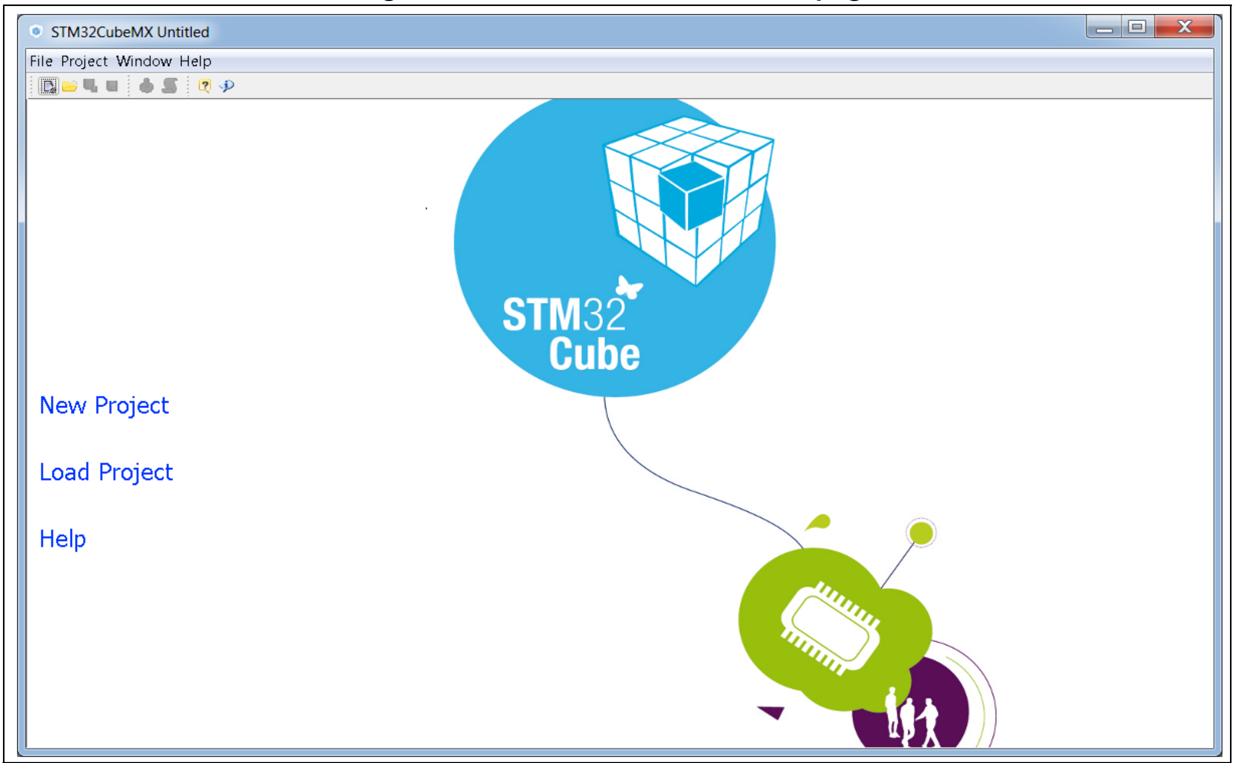


**Table 1. Welcome page shortcuts**

| Name | Description |
| --- | --- |
| New Project | Launches STM32CubeMX new project creation by opening the New project window (select an MCU from the MCU selector tab or a board configuration from the Board selector tab). |
| Load Project | Opens a browser window to select a previously saved configuration (.ioc file) and loads it.<br>When upgrading to a new version of STM32CubeMX, make sure to always backup your projects before loading the project (especially when the project includes user code). |
| Help | Opens the user manual. |

## 4.2 New project window

This window shows two tabs to choose from:

- The MCU selector tab offering a list of target processors
- A Board Selector tab showing a list of STMicroelectronics boards.

The MCU selector allows filtering on various criteria: series, lines, packages, peripherals and additional MCU characteristics such as memory size or number of I/Os (see *Figure 15*).

The Board selector allows filtering on STM32 board types, series and peripherals (see *Figure 16*).

When a board is selected, the **Pinout** view is initialized with the board default MCU and pinout configuration (see *Figure 18*). Optionally, the user can choose to initialize it with the default peripheral modes (see *Figure 19*).

When a board configuration is selected, the signals change to 'pinned', i.e. they cannot be moved automatically by STM32CubeMX constraint solver (user action on the peripheral tree, such as the selection of a peripheral mode, will not move the signals). This ensures that the user configuration remains compatible with the board.

**Figure 15. New Project window - MCU selector**

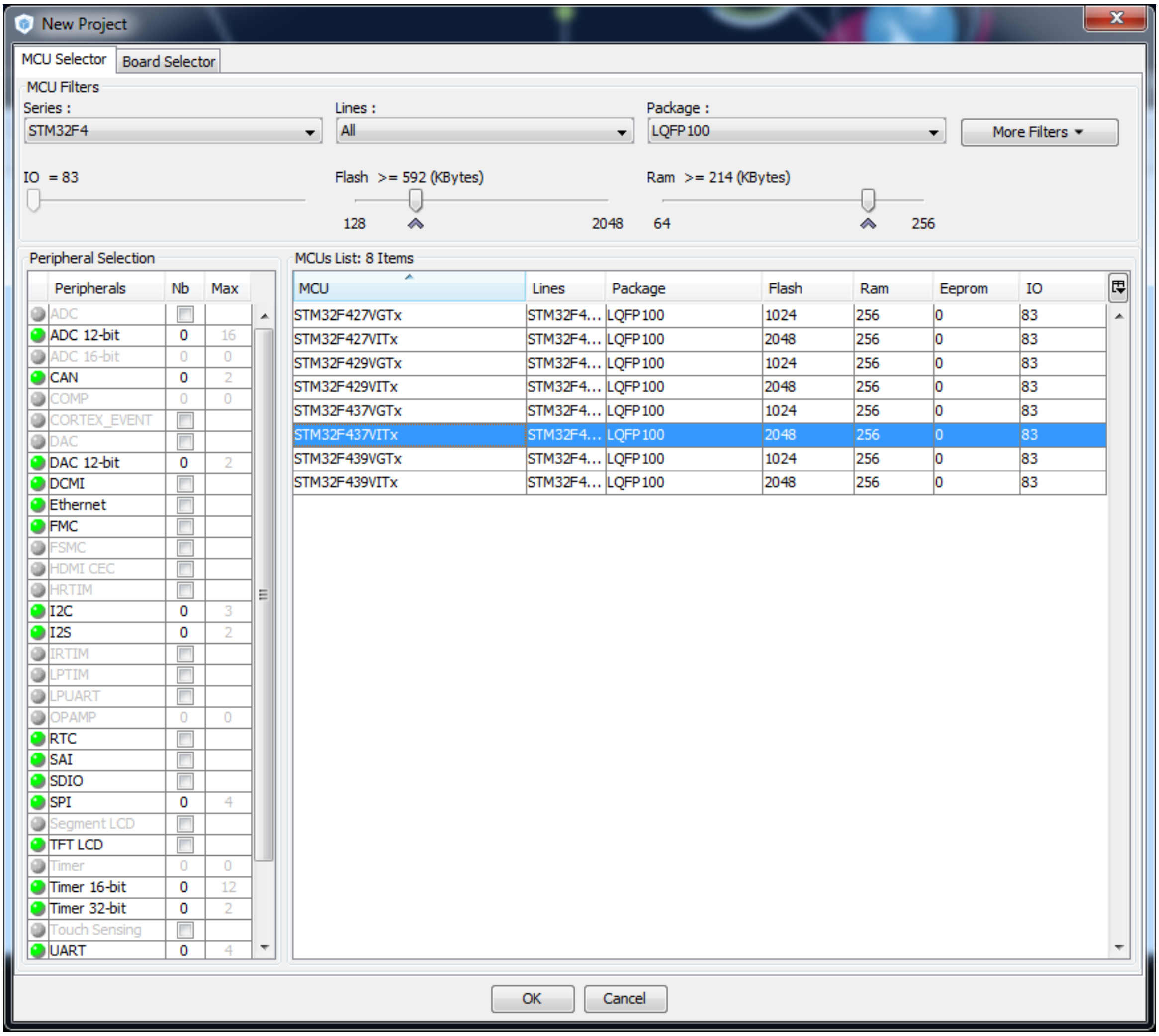
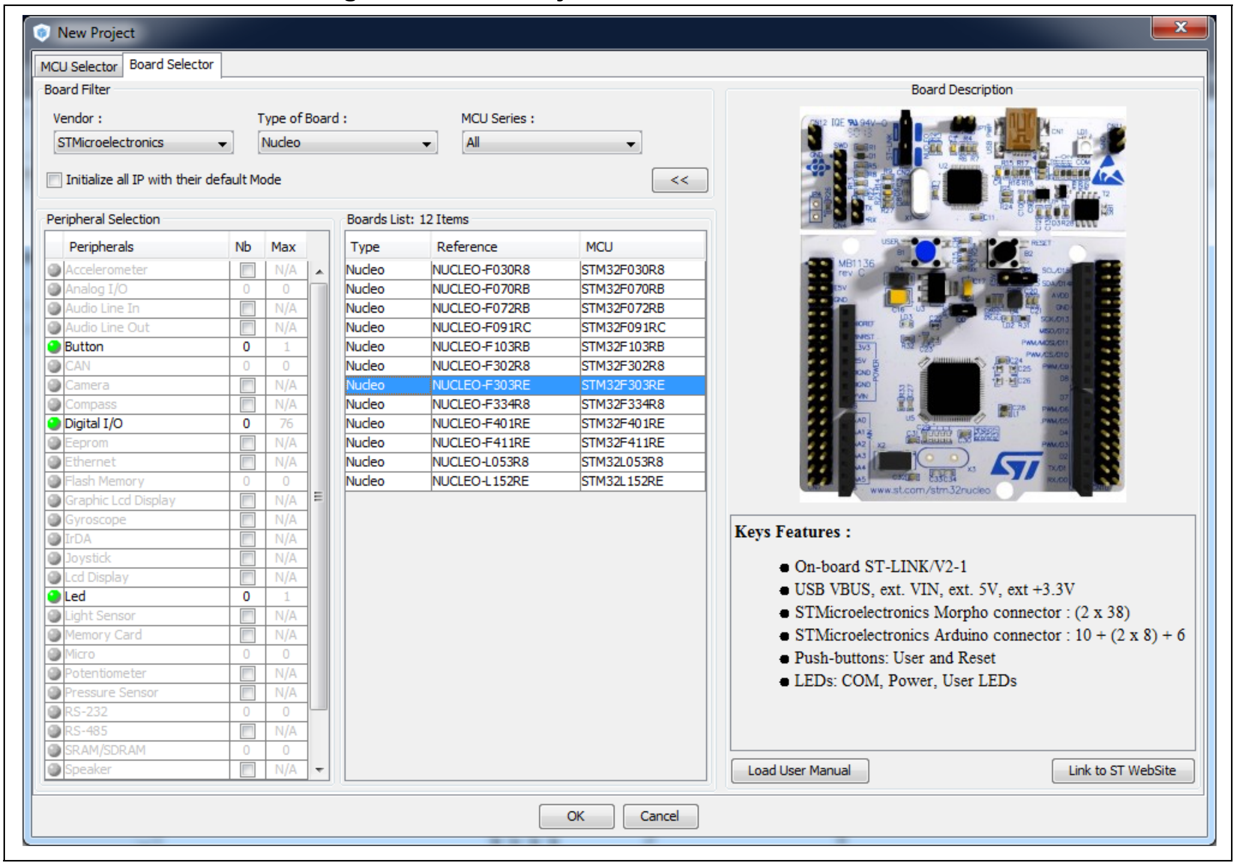


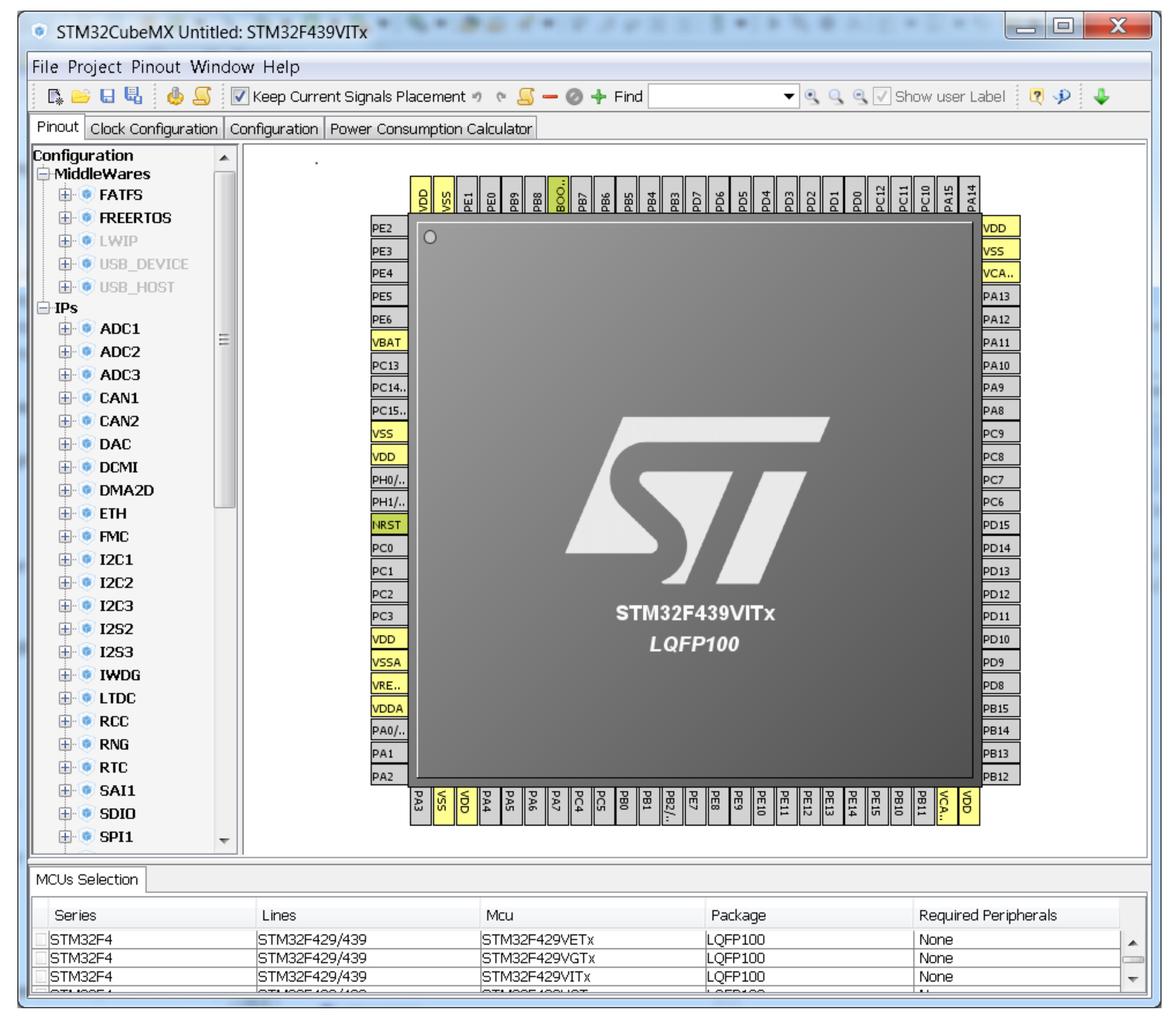
**Figure 16. New Project window - board selector**



## 4.3 Main window

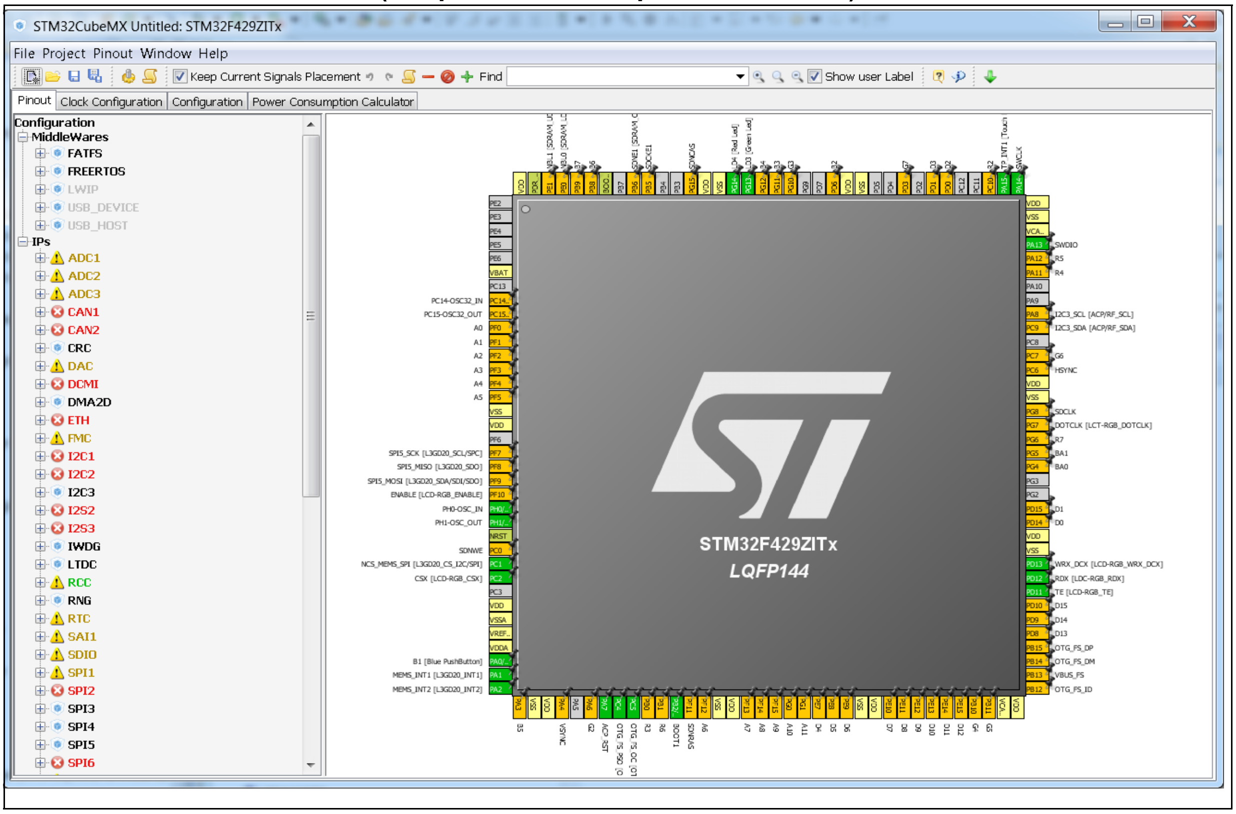
Once an STM32 part number or a board has been selected or a previously saved project has been loaded, the main window displays all STM32CubeMX components and menus (see *Figure 17*). Refer to *Section 4.3* for a detailed description of the toolbar and menus.

**Figure 17. STM32CubeMX Main window upon MCU selection**



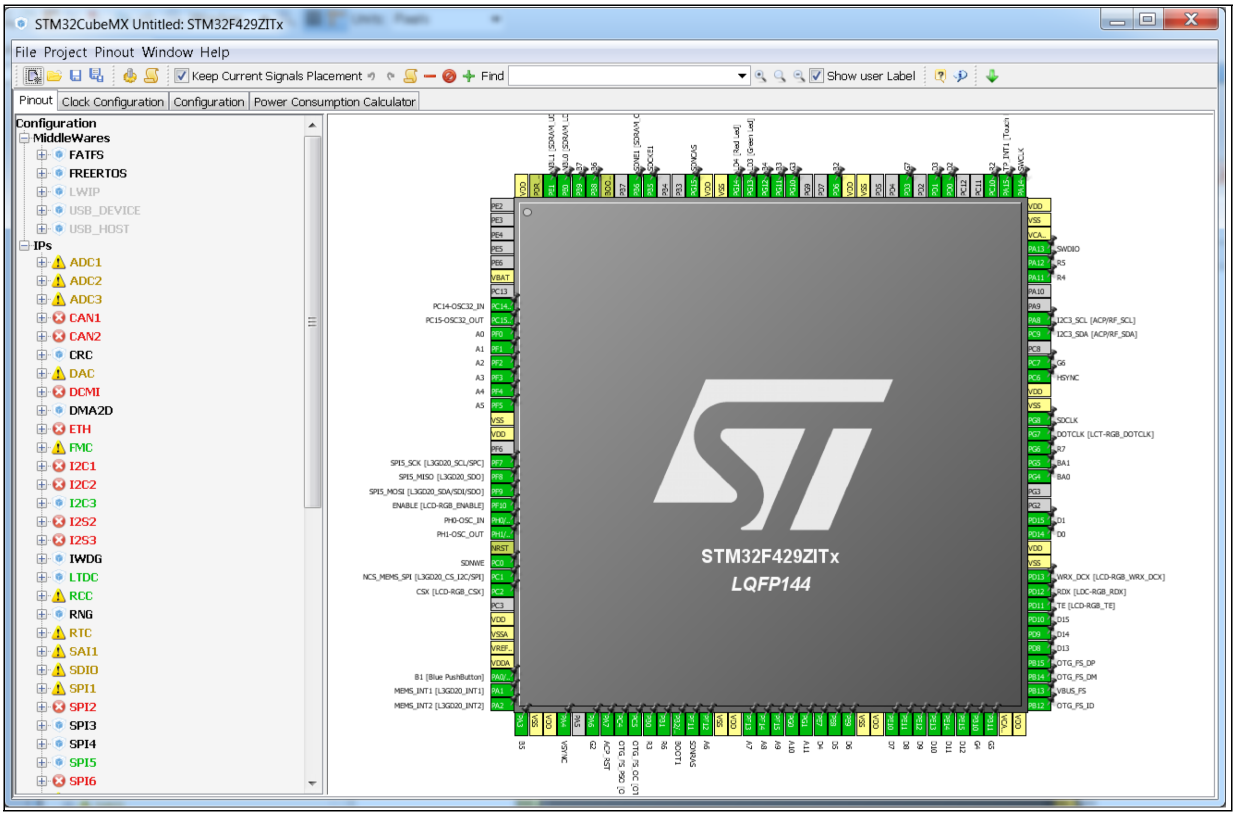
Selecting a board while keeping the peripheral default modes option unchecked, automatically sets the pinout for this board. However, no peripheral modes are set. The user can then manually select from the peripheral tree the peripheral modes required for his application (see *Figure 18*).

**Figure 18. STM32CubeMX Main window upon board selection (Peripheral default option unchecked)**



Selecting a board with the peripheral default modes option checked, automatically sets both the pinout and the default modes for the peripherals available on the board. This means that STM32CubeMX will generate the C initialization code for all the peripherals available on the board and not only for those relevant to the user application (see *Figure 19*).

**Figure 19. STM32CubeMX Main window upon board selection (Peripheral default option checked)**




## 4.4 Toolbar and menus

The following menus are available from STM32CubeMX menu bar:

- **File** menu
- **Project** menu
- **Pinout** menu (displayed only when the **Pinout** view has been selected)
- **Window** menu
- **Help** menu

STM32CubeMX menus and toolbars are described in the sections below.

### 4.4.1 File menu

Refer to *Table 2* for a description of the **File** menu and icons.

**Table 2. File menu functions**

| Icon | Name | Description |
|---|---|---|
|  | New Project | Opens a new project window showing all supported MCUs and well as a set of STMicroelectronics boards to choose from |
|  | Load Project … | Loads an existing STM32CubeMX project configuration by selecting an STM32CubeMX configuration .ioc file. |
|  | Save Project as … | Saves current project configuration (pinout, clock tree, IP, PCC) as a new project. This action creates an .ioc file with user defined name and located in the destination folder |
|  | Save Project | Saves current project |
| No icon | Close Project | Closes current project and switch back to the welcome page |
| No icon | Recent Projects > | Displays the list of five most recently saved projects |
| No icon | Exit | Proposes to save the project if needed then close the application |

### 4.4.2 Project menu

Refer to *Table 3* for a description of the **Project** menu and icons.

**Table 3. Project menu**

| Icon | Name | Description |
|---|---|---|
|  | Generate Code | Generates C initialization C code for current configuration (pinout, clocks, peripherals and middleware). Opens a window for project settings if they have not been defined previously. |
|  | Generate Report(1) | Generates current project configuration as a pdf file and a text file. |
|  | Settings | Opens the project settings window to configure project name, folder, select a toolchain and C code generation options |

1. If the project was previously saved, the reports are generated at the same location as the project configuration .ioc file. Otherwise, the user can choose the destination folder, and whether to save the project configuration as an .ioc file or not.

### 4.4.3 Pinout menu

The **Pinout** menu and sub-menus shortcuts are available only when the **Pinout** tab is selected (see *Figure 20*). They are hidden otherwise (see *Figure 21*). Refer to *Table 4* for a description of the **Pinout** menu and icons.

**Figure 20. Pinout menus (Pinout tab selected)**

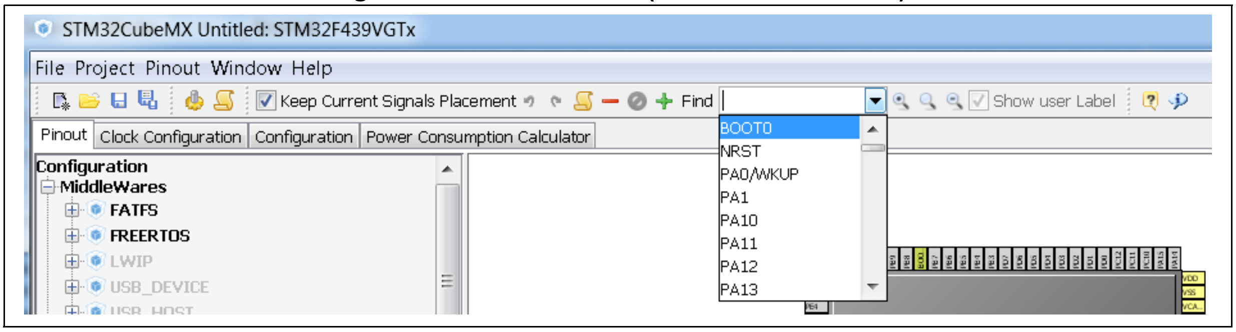



**Figure 21. Pinout menus (Pinout tab not selected)**

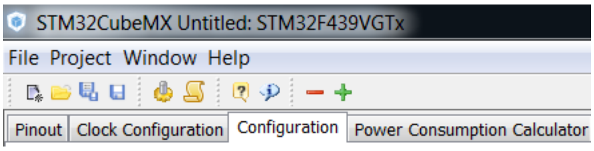



**Table 4. Pinout menu**

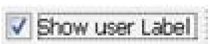
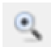
| Icon | Name | Description |
|---|---|---|
|  | Undo | Undoes last configuration steps (one by one) |
|  | Redo | Redoes steps that have been undone (one by one) |
| No icon | Pins/Signals Options | Opens a window showing the list of all the configured pins together with the name of the signal on the pin and a Label field allowing the user to specify a label name for each pin of the list. For this menu to be active, at least one pin must have been configured.<br>Click the pin icon to pin/unpin signals individually.<br>Select multiple rows then right click to open contextual menu and select action to pin or unpin all selected signals at once.<br>Click column header names to sort alphabetically by name or according to placement on MCU. |
|   | Pinout search field | Allows the user to search for a pin name, signal name or signal label in the **Pinout** view. When it is found, the pin or set of pins that matches the search criteria blinks on the **Chip** view. Click the **Chip** view to stop blinking. |
|   | Show user labels | Allows showing on the **Chip** view, the user-defined labels instead of the names of the signals assigned to the pins. |

**Table 4. Pinout menu (continued)**

| Icon | Name | Description |
|---|---|---|
| No icon | Clear Pinouts | Clears user pinout configuration in the Pinout window. Note that this action clears from the configuration window the IPs that have an influence on the pinout. |
| No icon | Clear Single Mapped Signals | Clears signal assignments to pins for signals that have no associated mode (highlighted in orange and not pinned). |
| No icon | Set unused GPIOs | Opens a window to specify the number of GPIOs to be configure among the total number of GPIO pins that are not used yet. Specify their mode: Input, Output or Analog (recommended configuration to optimize power consumption). |
| No icon | Reset used GPIOs | Opens a window to specify the number of GPIOs to be freed among the total number of GPIO pins that are configured. |
|  | Generate csv text pinout file | Generates pin configuration as a .csv text file |
|  | Collapse All | Collapses the IP / Middleware tree view |
|  | Disable Modes | Resets to “Disabled” all peripherals and middleware modes that have been enabled. The pins configured in these modes (green color) are consequently reset to “Unused” (gray color). IPs and middleware labels change from green to black (when unused) or gray (when not available). |
|  | Expand All | Expands the IP/Middleware tree view to display all functional modes. |
|  | Zooming in | Zooms in the chip pinout diagram |
|  | Best Fit | Adjusts the chip pinout diagram to the best fit size |
|  | Zooming out | Zooms out the chip pinout diagram |
|  Keep Current Signals Placement | Keep current signals Placement | Available from toolbar only.<br>Prevents moving pin assignments to match a new IP operating mode. It is recommended to use the new pinning feature that can block each pin assignment individually and leave this checkbox unchecked. |

## 4.4.4 Window menu

The **Window** menu allows to access the **Outputs** function (see *Table 5*).

**Table 5. Window menu**

| Name | Description |
|---|---|
| Outputs | Opens the MCUs selection window at the bottom of STM32CubeMX Main window. |

### 4.4.5 Help menu

Refer to *Table 6* for a description of the **Help** menu and icons.

**Table 6. Help menu**

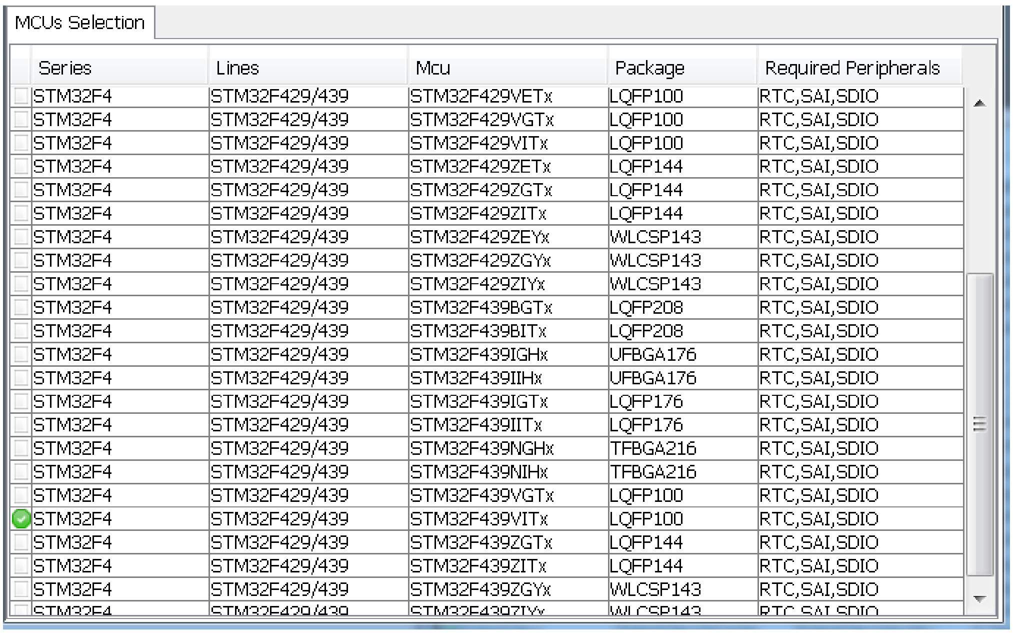
| Icons | Name | Description |
|---|---|---|
| | Help Content | Opens the STM32CubeMX user manual |
| | About... | Shows version information |
| | Check for Updates | Shows the software and firmware release updates available for download. |
| | Install New Libraries | Shows all STM32CubeMX and firmware releases available for installation. Green check box indicates which ones are already installed on you PC and up-to-date. |
| | Updater Settings... | Opens the updater settings window to configure manual versus automatic updates, proxy settings for internet connections, repository folder where the downloaded software and firmware releases will be stored. |

## 4.5 MCUs selection window

This window lists all the MCUs of a given family that match the user criteria (series, peripherals, package..) when an MCU was selected last.

*Note: Selecting a different MCU from the list resets the current project configuration and switches to the new MCU. The user will be prompted to confirm this action before proceeding.*

**Figure 22. MCU selection menu**



MCUs Selection

| Series | Lines | Mcu | Package | Required Peripherals |
|---|---|---|---|---|
| STM32F4 | STM32F429/439 | STM32F429VETx | LQFP100 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F429VGTx | LQFP100 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F429VITx | LQFP100 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F429ZETx | LQFP144 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F429ZGTx | LQFP144 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F429ZITx | LQFP144 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F429ZEYx | WLCSP143 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F429ZGYx | WLCSP143 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F429ZIYx | WLCSP143 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439BGTx | LQFP208 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439BITx | LQFP208 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439IGHx | UFBGA176 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439IIHx | UFBGA176 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439IGTx | LQFP176 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439IITx | LQFP176 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439NGHx | TFBGA216 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439NIHx | TFBGA216 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439VGTx | LQFP100 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439VITx | LQFP100 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439ZGTx | LQFP144 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439ZITx | LQFP144 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439ZGYx | WLCSP143 | RTC,SAI,SDIO |
| STM32F4 | STM32F429/439 | STM32F439ZIYx | WLCSP143 | RTC,SAI,SDIO |

This window can be shown/hidden by selecting/unselecting **Outputs** from the Window menu.

## 4.6 Set unused / Reset used GPIOs windows

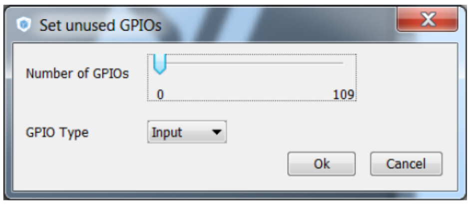
These windows allow configuring several pins at a time in the same GPIO mode.

To open them:

- Select **Pinout** > **Set unused GPIOs** from the STM32CubeMX menu bar.

*Note: The user selects the number of GPIOs and lets STM32CubeMX choose the actual pins to be configured or reset, among the available ones.*

**Figure 23. Set unused pins window**




- Select **Pinout > Reset used GPIOs** from the STM32CubeMX menu bar.

  Depending whether the Keep Current Signals Placement option is checked or not on the toolbar, STM32CubeMX conflict solver will be able to move or not the GPIO signals to other unused GPIOs:
  - When Keep Current Signals Placement is off (unchecked), STM32CubeMX conflict solver can move the GPIO signals to unused pins in order to fit in another peripheral mode.
  - When Keep Current Signals Placement is on (checked), GPIO signals will not be moved and the number of possible peripheral modes becomes limited.

  Refer to *Figure 25* and *Figure 26* and check the limitation in available peripheral modes.

**Figure 24. Reset used pins window**

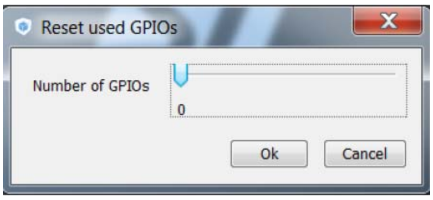


**Figure 25. Set unused GPIO pins with Keep Current Signals Placement checked**

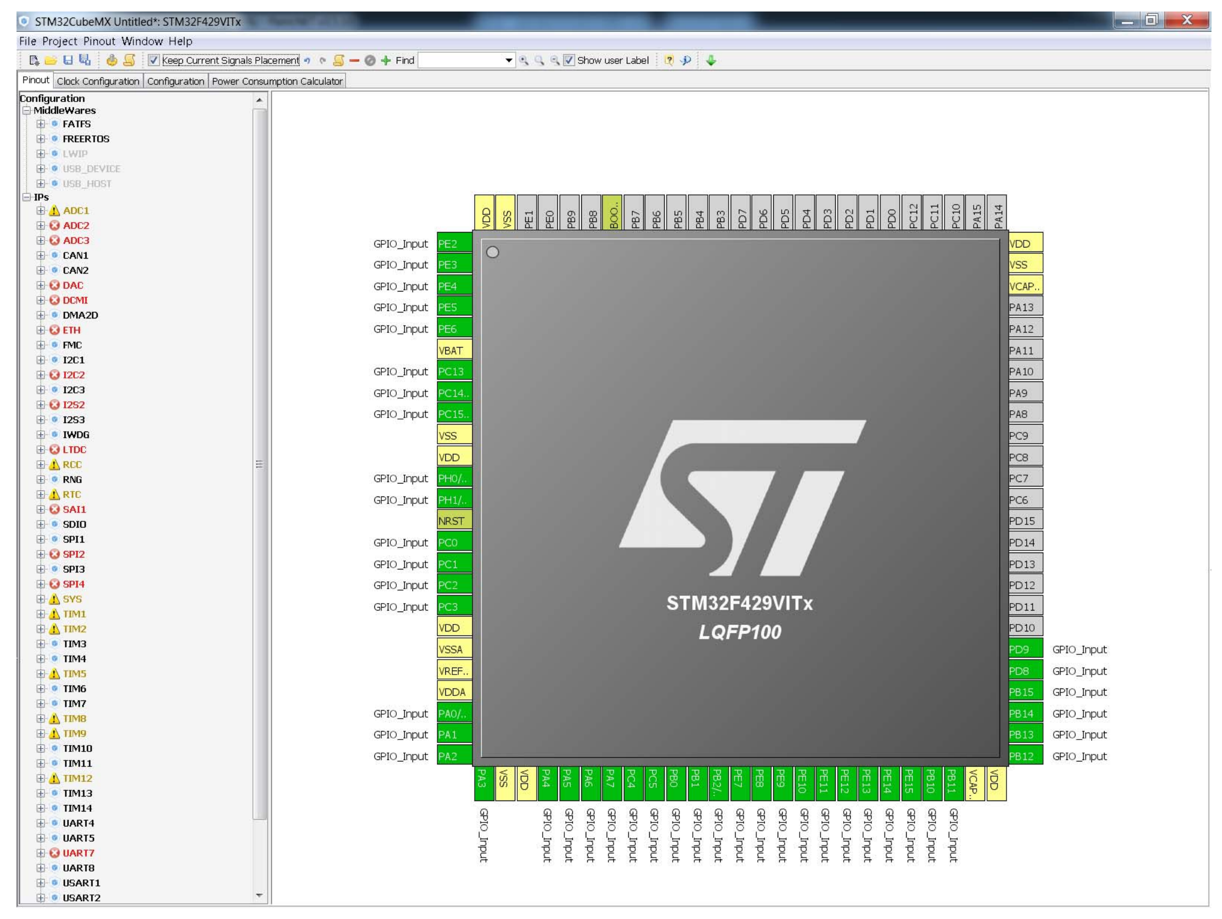
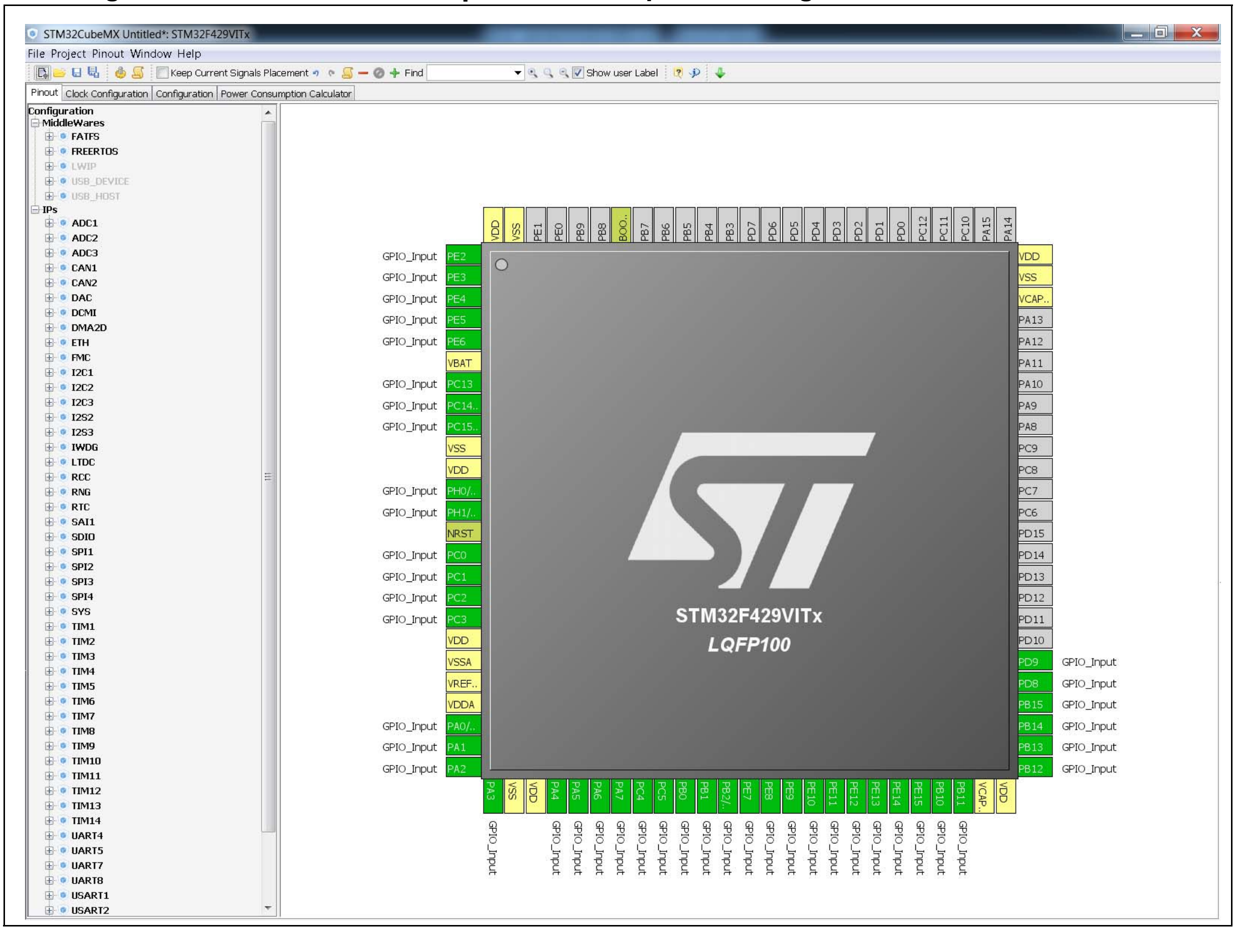


**Figure 26. Set unused GPIO pins with Keep Current Signals Placement unchecked**




## 4.7 Project Settings Window

This window allows configuring the project: project name, project location, choice of Integrated Development Environment tools (Keil MDK-ARM, IAR EW-ARM, Attolic TrueStudio, AC6 SW4STM32 …), and C code generation options.

There are several ways to enter project settings information:

1. By selecting **Project > Settings** from the STM32CubeMX menu bar (see *Figure 27*). The code generation will then be generated in the project folder tree shown in *Figure 28*.

**Figure 27. Project Settings window**

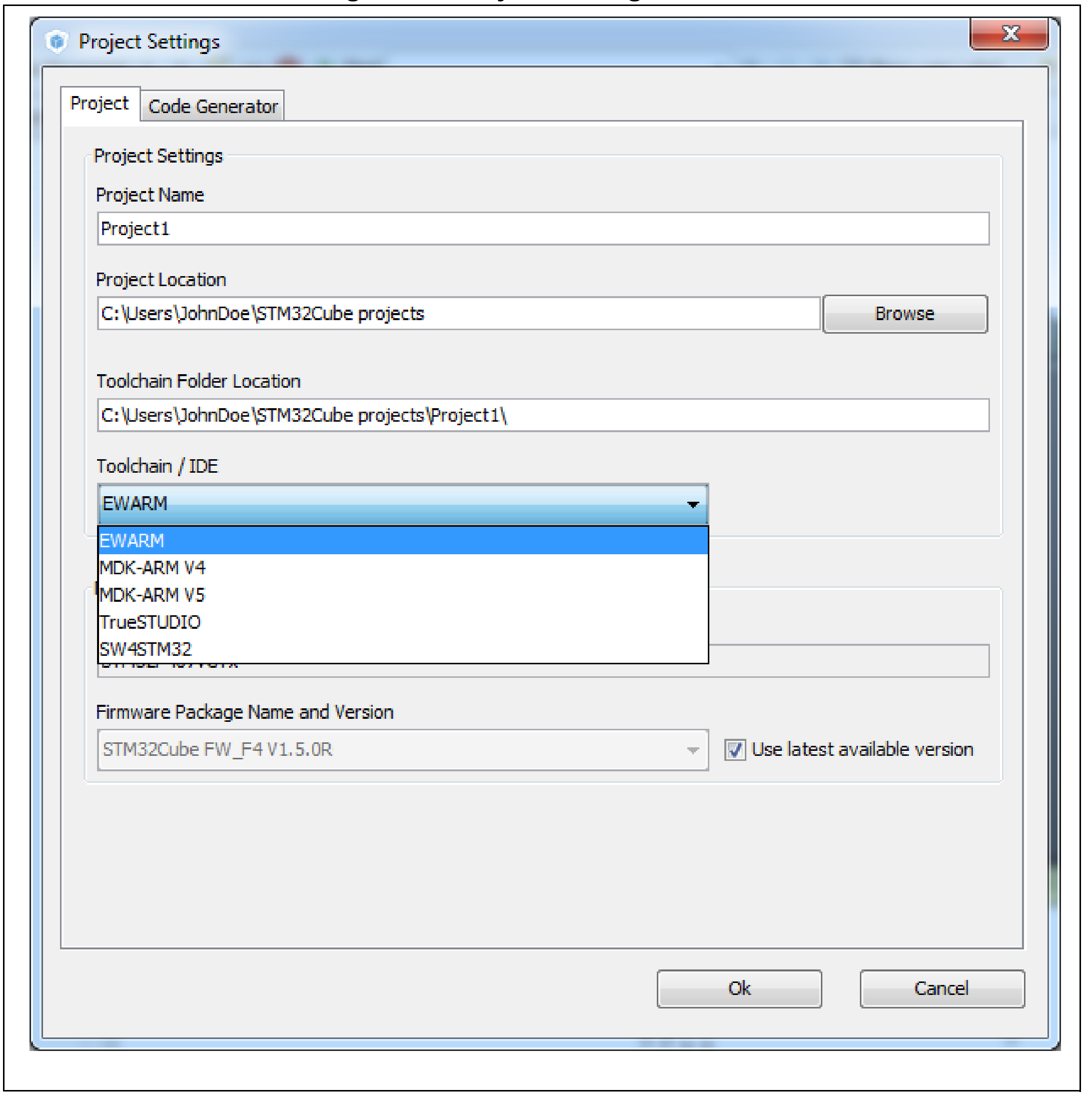
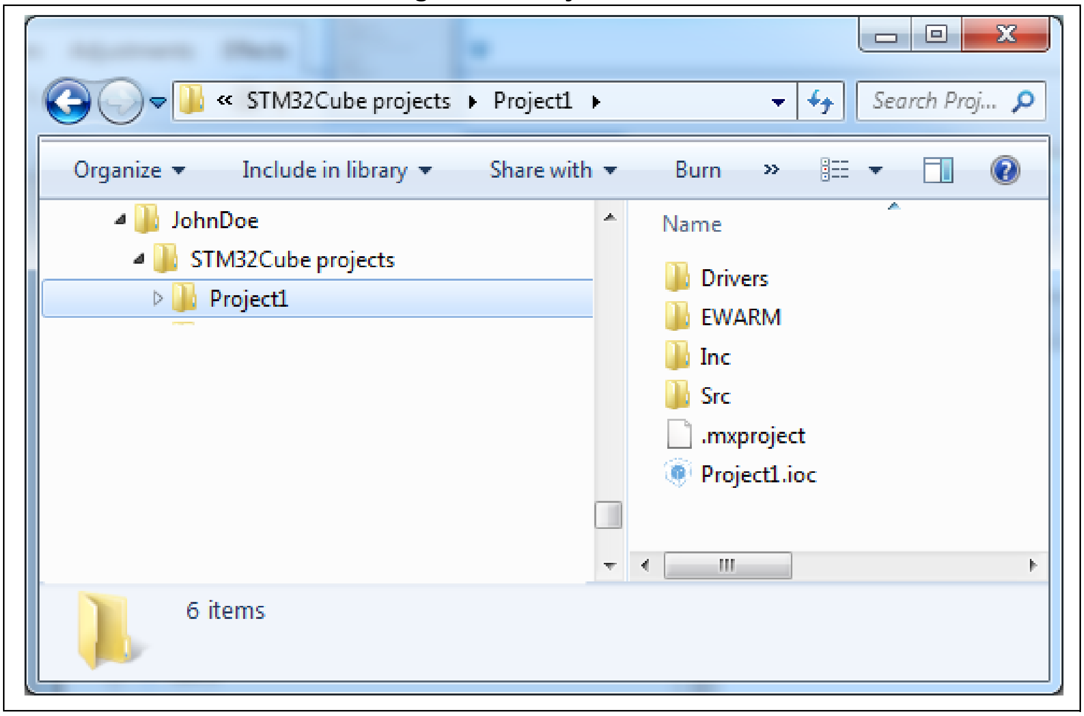
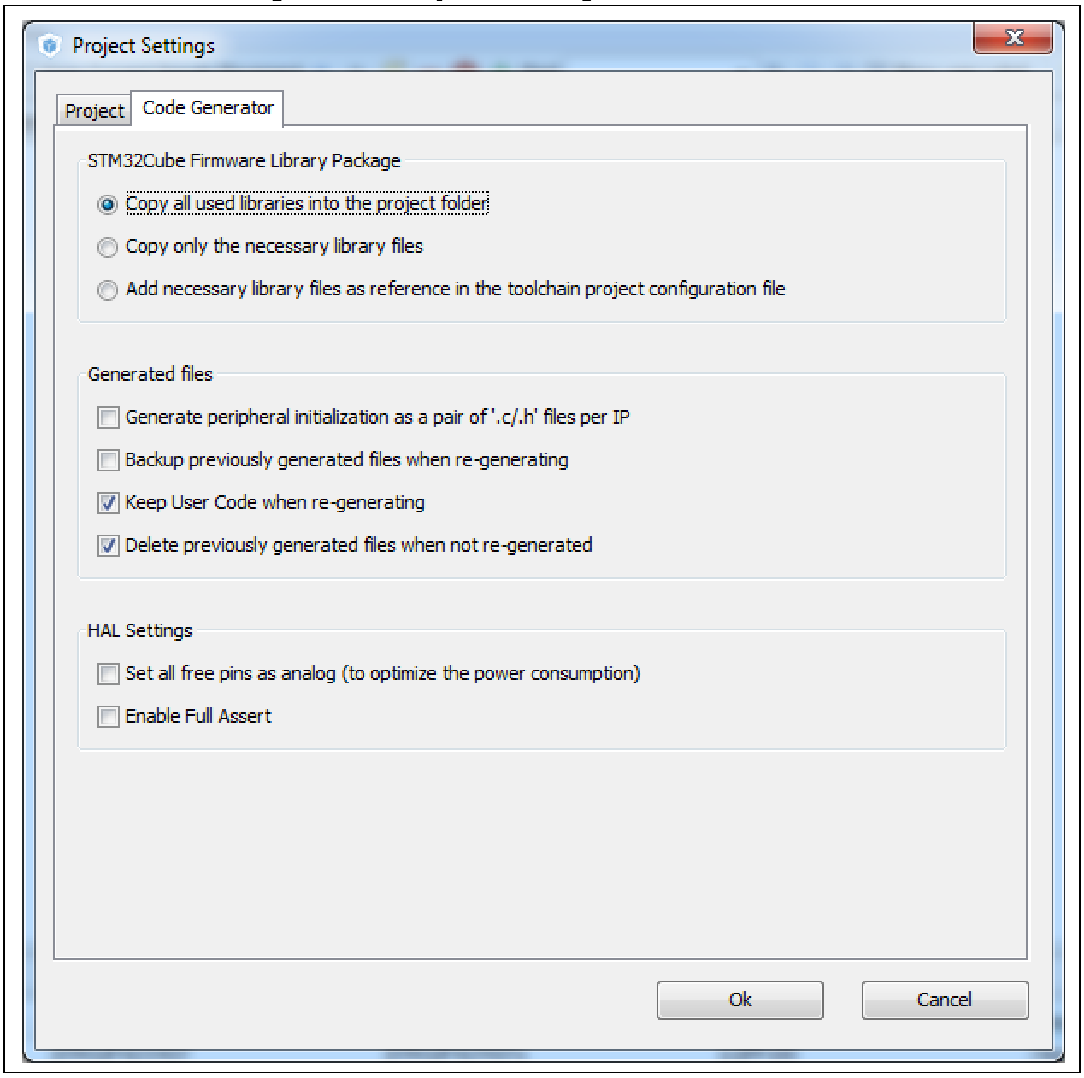


**Figure 28. Project folder**



2. By clicking **Project** > **Generate code** for the first time (see *Figure 29*).

**Figure 29. Project Settings Code Generator**




3. By selecting **Save As** for a project that includes C code generation (and not only pin configuration).

   Select the Code Generator tab to specify the following code generation options.

   – Copy all necessary 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.

HAL settings options are also offers:

– For the generated initialization code to set all free pins as analog for power consumption optimization

– To use or not the "Full Assert" function: the Define statement in the stm32xx_hal_conf.h configuration file will be commented or uncommented, respectively.

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

## 4.8 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.
3. Select **Help > Update settings** to open the **Updater settings** window and configure update mechanism settings (proxy settings, manual versus automatic updates).

## 4.9 About Window

This window displays STM32CubeMX version information.

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

**Figure 30. About window**

## 4.10 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 an 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_sdio.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:

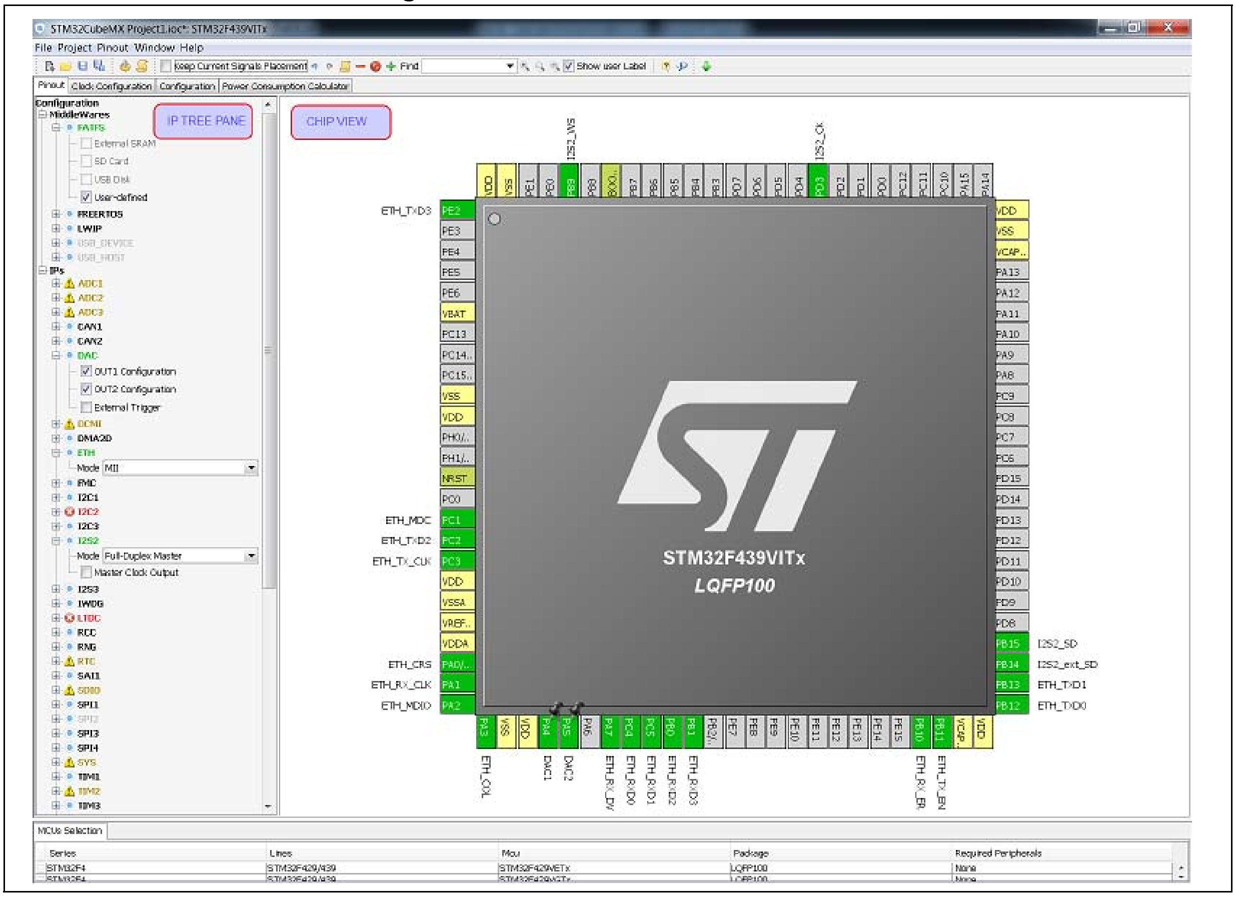
- From the **IP tree** by clicking the peripheral names and selecting the operating modes (see *Section 4.10.1: IP tree pane*).
- For advanced users, by clicking a pin on the **Chip** view to manually map it to a peripheral function (see *Section 4.10.2: Chip view*).

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 31. STM32CubeMX Pinout view**




## 4.10.1 IP tree pane

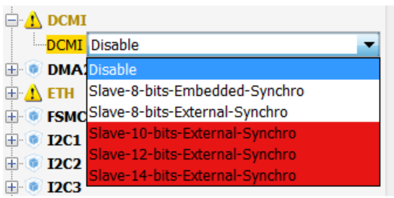
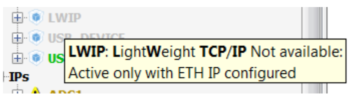
In this pane, the user can select the peripherals, services (DMA, RCC,...), middleware in the modes corresponding to the application.

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

**Table 7. 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<br>Mode Disable | Available peripheral mode configurations are shown in plain black. |
|  | The warning yellow icon indicates that at least one mode configuration is no longer available. |
|  | When no more configurations are left for a given peripheral mode, this peripheral is highlighted in red. |
|  | Some modes depends on the configuration of other peripherals or middleware modes. A tooltip explains the dependencies when the conditions are not fulfilled. |

## 4.10.2 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 32* for an example).

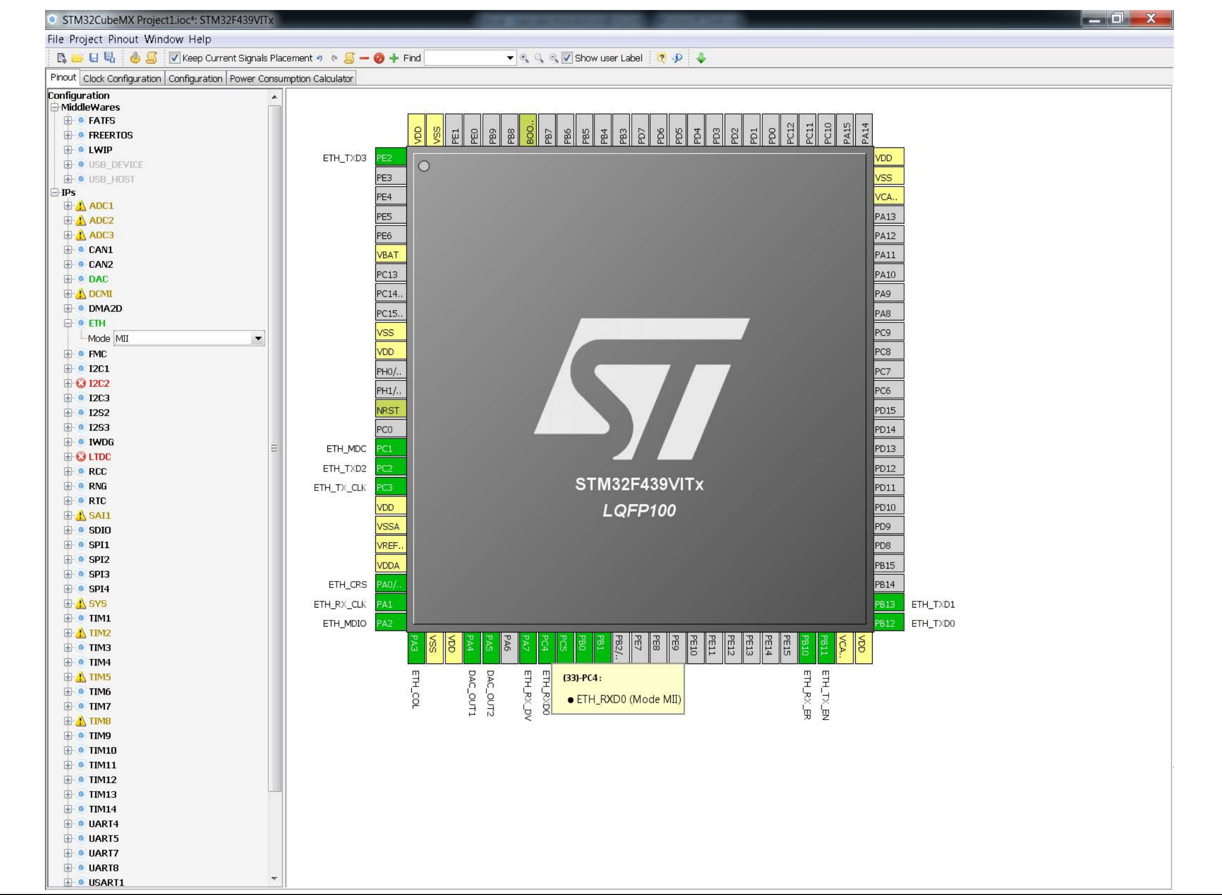
The **Chip** view is automatically refreshed to match the user configuration performed via the peripheral tree. It shows the pins current configuration state.

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 4*).
- 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 32. Chip view**



### Icons and color schemes

*Table 8* shows the icons and color scheme used in the **Chip** view.

**Table 8. STM32CubeMX Chip view - Icons and color scheme**

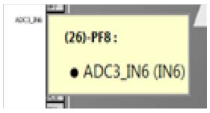
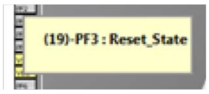
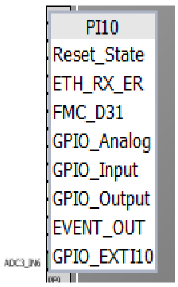
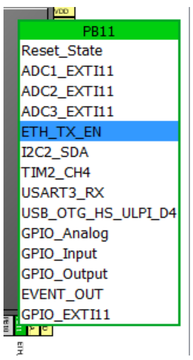
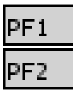
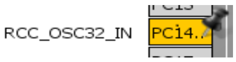
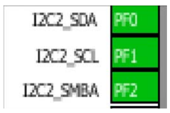
| Display | Pin information |
|---|---|
|       | **Tooltip** indicates the selected pin current configuration: alternate function name, Reset state or GPIO mode.<br>Move your mouse over the pin name to display it.<br>When a pin features alternate pins corresponding to the function currently selected, a popup message prompts the user to perform a ctrl + click to display them.<br>The alternate pins available are highlighted in blue. |
|   | **List** of alternate functions that can be selected for a given pin. By default, no alternate function is configured (pin in reset state).<br>Click the pin name to display the list. |
|   | When a function has been mapped to the pin, it is highlighted in blue.<br>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 8. STM32CubeMX Chip view - Icons and color scheme (continued)**

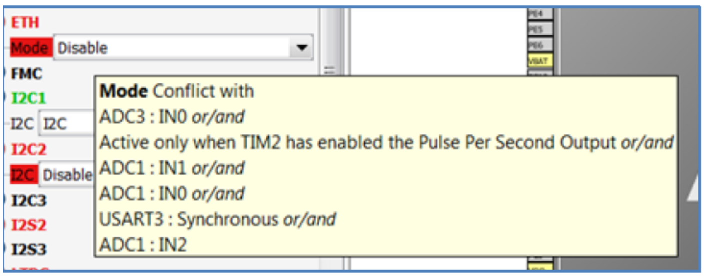
| Display | Pin information |
|---|---|
|   | Power dedicated pins are highlighted in yellow. Their configuration cannot be changed. |
|   | Non-configured pins are shown in gray (default state). |
|   | When a signal assignment corresponds to a peripheral mode without ambiguity, the pin color switches 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.<br>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.

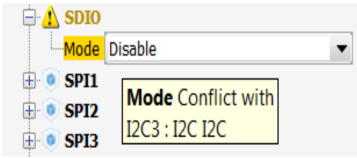
As an example (see *Figure 33*), the Ethernet (ETH) peripheral is no longer available because there is no possible mode configuration left. A tooltip indicates to which signal are assigned the pins required for this mode (ADC1-IN0 signal, USART3 synchronous signal, etc...).

**Figure 33. Red highlights and tooltip example: no mode configuration available**



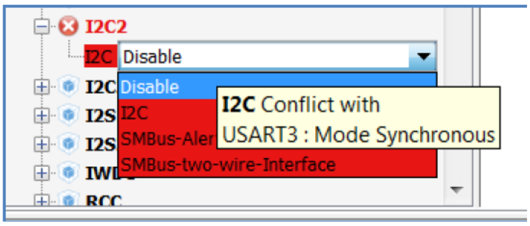

In the next example (see *Figure 34*), the SDIO peripheral is partially available because at least one of its modes is unavailable: the necessary pins are already assigned to the I2C mode of the I2C3 peripheral.

**Figure 34. Orange highlight and tooltip example: some configurations unavailable**




In this last example (see *Figure 35*) 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 35. Tooltip example: all configurations unavailable**



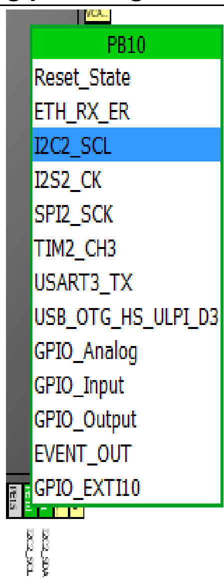

## 4.10.3 Chip view advanced actions

### 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 36*).
2. Click to select the new function to assign to the pin.

**Figure 36. 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.

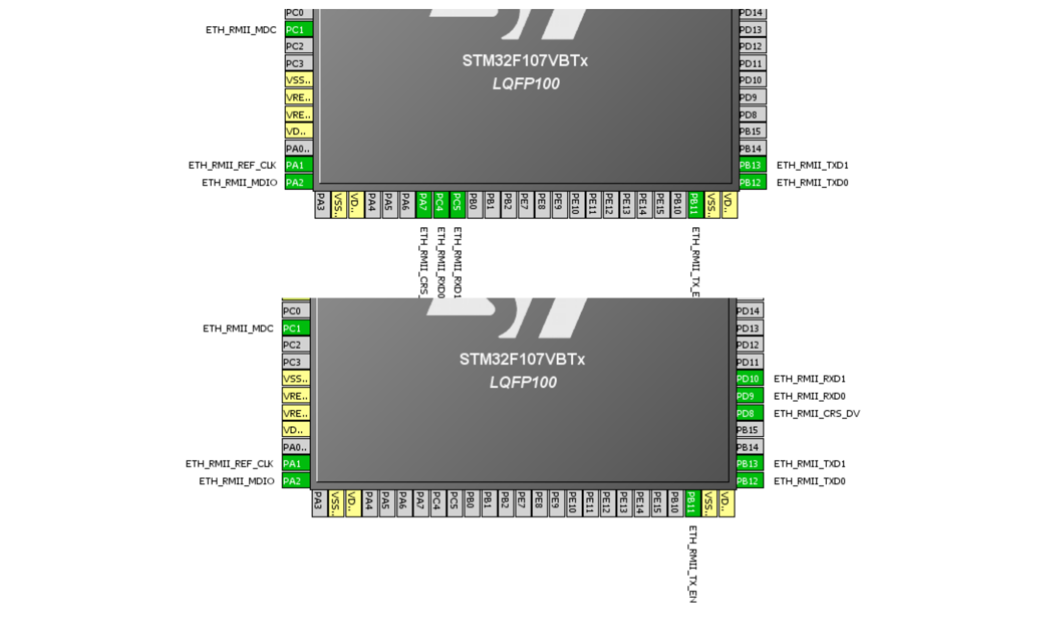
**Caution:** A pin assignment performed from the Chip view overwrites any previous assignment.

### 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 37, 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 37. 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.

### 4.10.4 Keep Current Signals Placement

This checkbox is available from the toolbar when the **Pinout** view is selected (see *Figure 20* and *Table 4*). 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.*

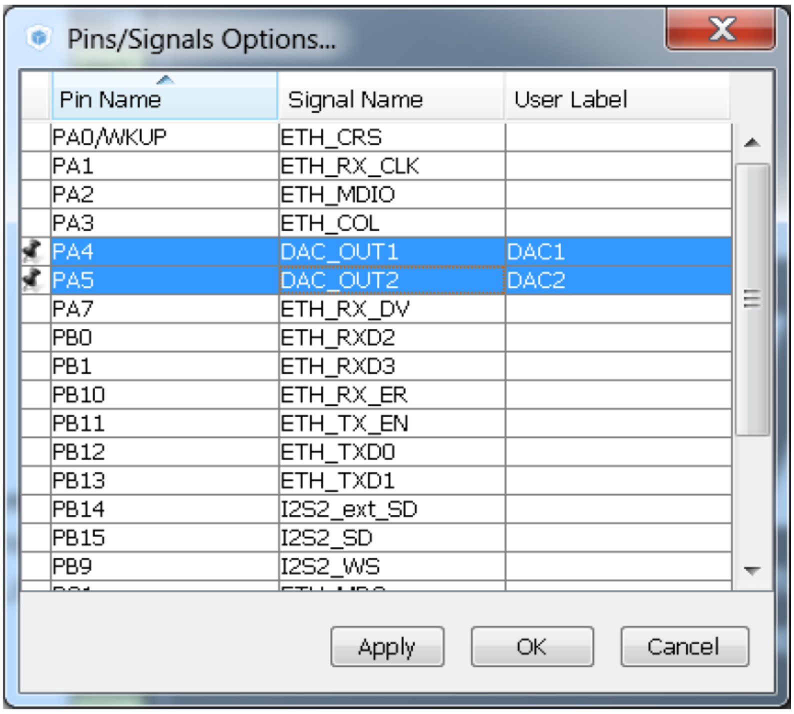
### 4.10.5 Pinning and labeling signals on pins

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 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 38*) 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 38. Pins/Signals Options window**




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 Configuration view

The **Configuration** view is available from the STM32CubeMX menu (see *Figure 17*). It helps configuring the GPIO parameters and the IP/middleware operating modes in order to generate the initialization C code.

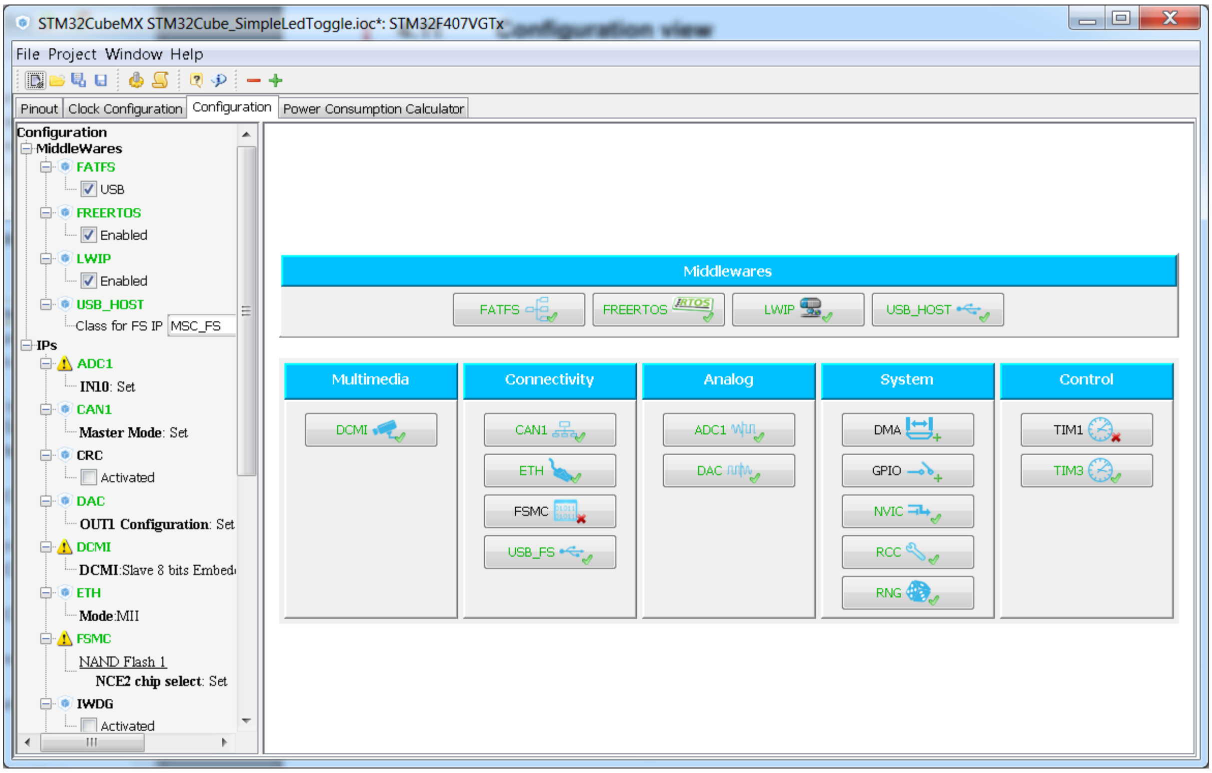
*Note: GPIO and Peripheral modes that influence the pinout can be set only from the* ***Pinout*** *view. They are read-only in the Configuration view.*

*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 an 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_sdio.c/.h driver files with the necessary code.*

In this view, the MCU is shown on the left pane by its IP tree and on the right pane, by the list of IPs organized in Middleware, Multimedia, Connectivity, Analog, System and Control categories. Each IP instance has a dedicated button to edit its configuration: as an example, TIM1 and TIM3 TIM instances are shown as dedicated buttons in *Figure 39*.

**Figure 39. STM32CubeMX Configuration view**




An IP configuration button is associated to each peripheral in the **Configuration** window (see *Table 9*).

**Table 9. IP configuration buttons**

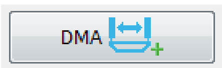
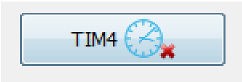
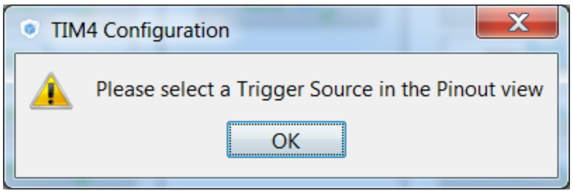
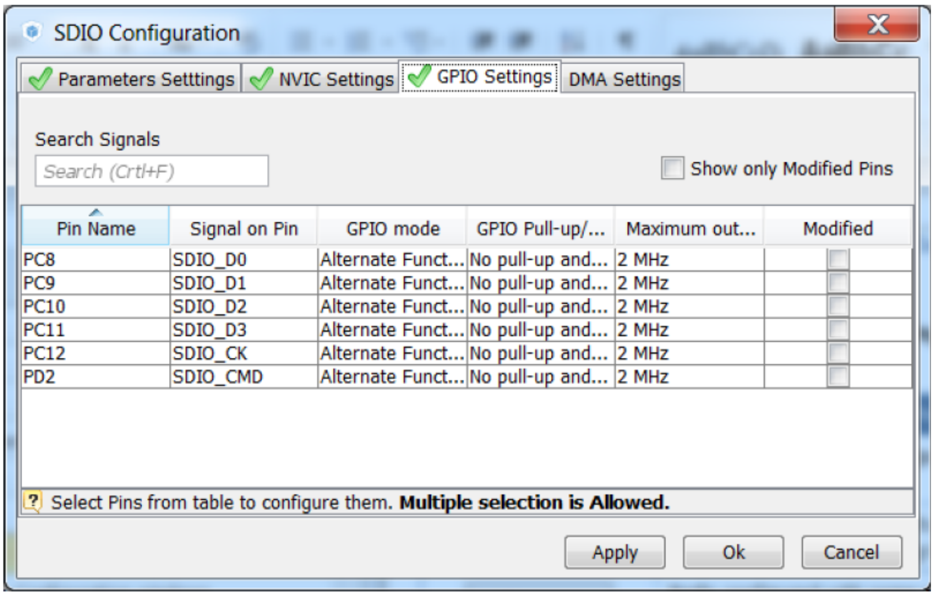
| Format | Peripheral Instance configuration status |
|---|---|
|  | Available but not fully configured yet. Click to open the configuration window. |
|  | Well configured with default or user-defined settings that allows proceeding with the generation of corresponding initialization C code. Click to open the configuration window. |

**Table 9. IP configuration buttons (continued)**

| | |
|---|---|
|   | Badly configured with some wrong parameter values. Click to display the errors highlighted in red.<br>Other example (UART):<br>  |
|   | Dialog box that explains source of error. It shall be fixed in another view. |

**GPIO, DMA and NVIC** settings can be accessed either via a dedicated button like other IPs or via a tab in the other configuration windows of the IPs which use them (see *Figure 40*).

**Figure 40. Configuration window tabs for GPIO, DMA and NVIC settings (STM32F4 series)**




## 4.11.1 IP and Middleware Configuration window

This window is open by clicking the IP instance or Middleware name from the **Configuration** pane. It allows to configure the functional parameters that are required for initializing the IP or the middleware in the selected operating mode. This configuration is used to generate the corresponding initialization C code. Refer to *Figure 41* for an IP Configuration windows example.

Invalid settings are detected and are either:

- Reset to minimum valid value if user’s choice was smaller than minimum threshold,
- Reset to maximum valid value if user’s choice was greater than maximum threshold,
- Reset to previous valid value if previous value was neither a maximum nor a minimum threshold value,
- Highlighted in red: ✖ 1000000 Bits/s

*Table 9* describes IP and middleware configuration buttons and messages.

**Figure 41. UART4 IP Configuration window (STM32F4 series)**

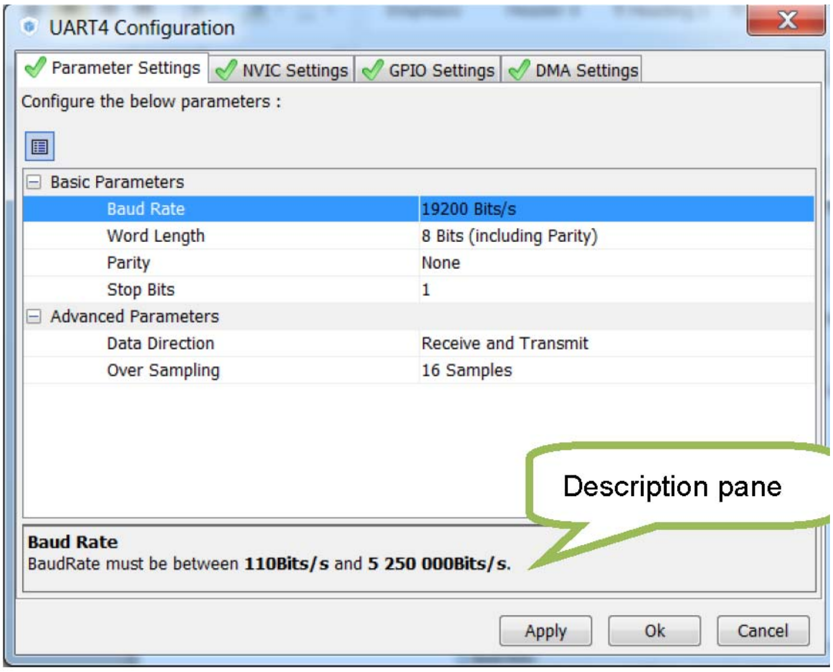
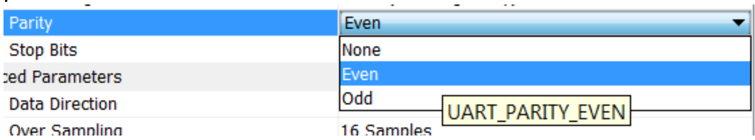
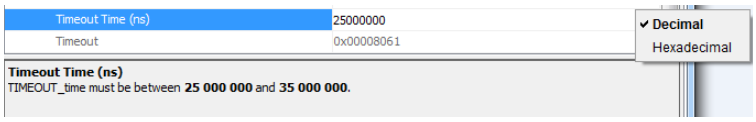
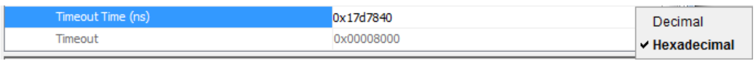



**Table 10. IP Configuration window buttons and tooltips**

| Buttons and messages | Action |
|---|---|
| Apply | Saves the changes without closing the window |
| OK | Saves and closes the window |
| Cancel | Closes and resets previously saved parameter settings |
|  | Shows and Hides the description pane |

**Table 10. IP Configuration window buttons and tooltips (continued)**

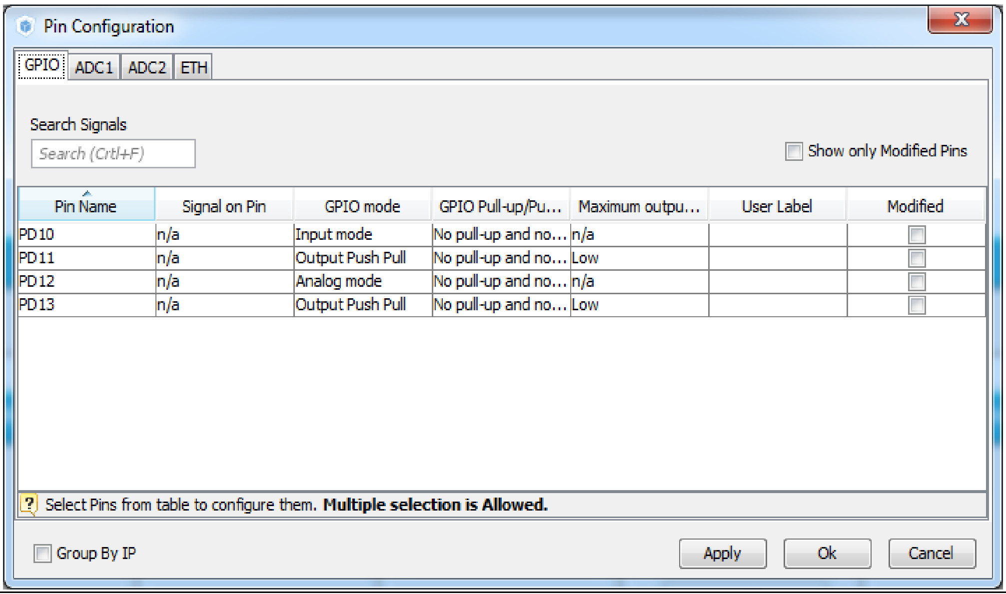
| Buttons and messages | Action |
|---|---|
| Tooltip | Guides the user through the settings of parameters with valid min-max range.<br>To display it, moves the mouse over a parameter value from a list of possible values.<br><br> |
| Hexadecimal vs decimal values | Choose to display the field as an hexadecimal or a decimal value by clicking the arrow on the right:<br><br><br><br> |

## 4.11.2 GPIO Configuration window

Click **GPIO** in the **Configuration** pane to open the **GPIO configuration** window that allows to configure the settings of the GPIO pins (see *Figure 42*).

*Note: It is also possible to access GPIO settings for a specific IP instance via the dedicated GPIO tab in the IP instance configuration window.*

**Figure 42. GPIO Configuration window - GPIO selection**



Click a row or select a set of rows to display the corresponding GPIO parameters (see *Figure 43*):

- **GPIO mode** (analog, input, output, alternate function)

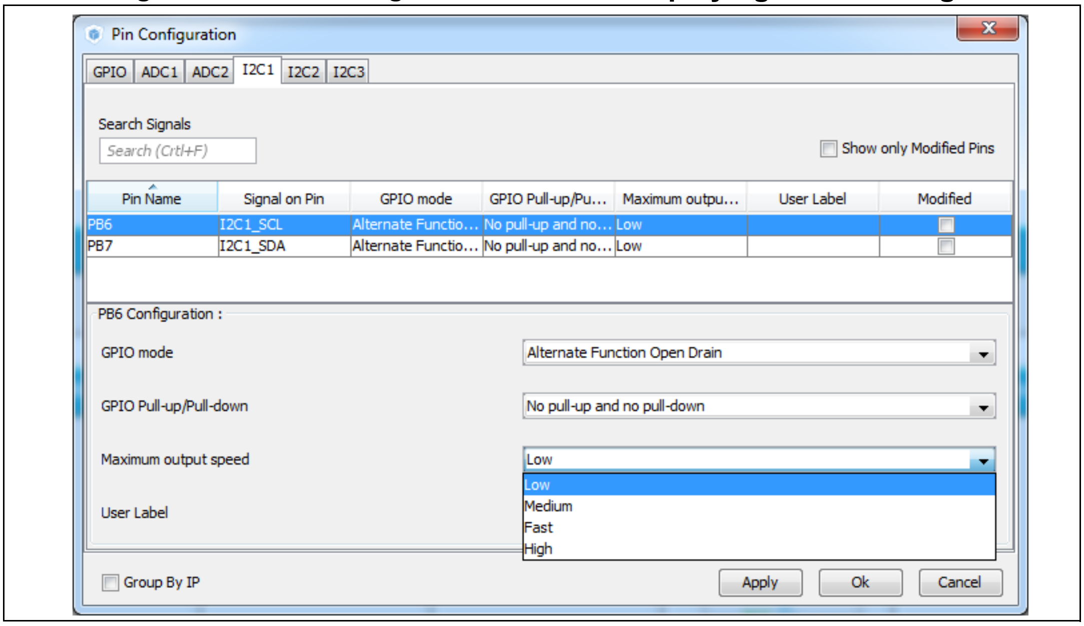
  Selecting an IP mode in the **Pinout** view automatically configures the pins with the relevant alternate function and GPIO mode.
- **GPIO pull-up/pull-down**

  It is set to a default value and can be configured when other choices are possible.
- **GPIO maximum output speed** (for communication IPs only)

  It is set to Low by default for power consumption optimization and can be changed to a higher frequency to fit application requirements.
- **User Label**

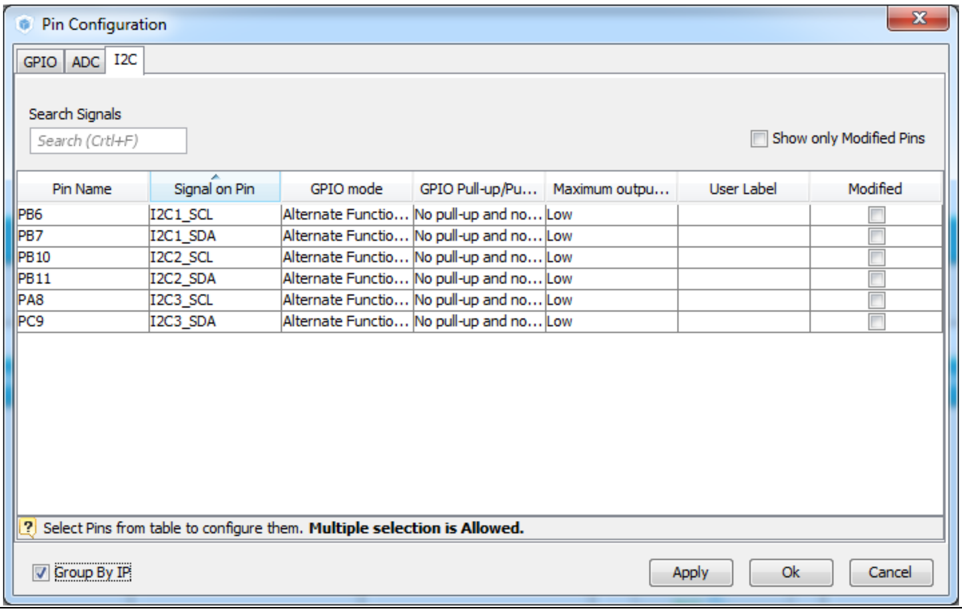
  It changes the default name (e.g. GPIO_input) into a user defined name. The **Chip** view is updated accordingly. The GPIO can be found under this new name via the Find menu.

**Figure 43. GPIO Configuration window - displaying GPIO settings**



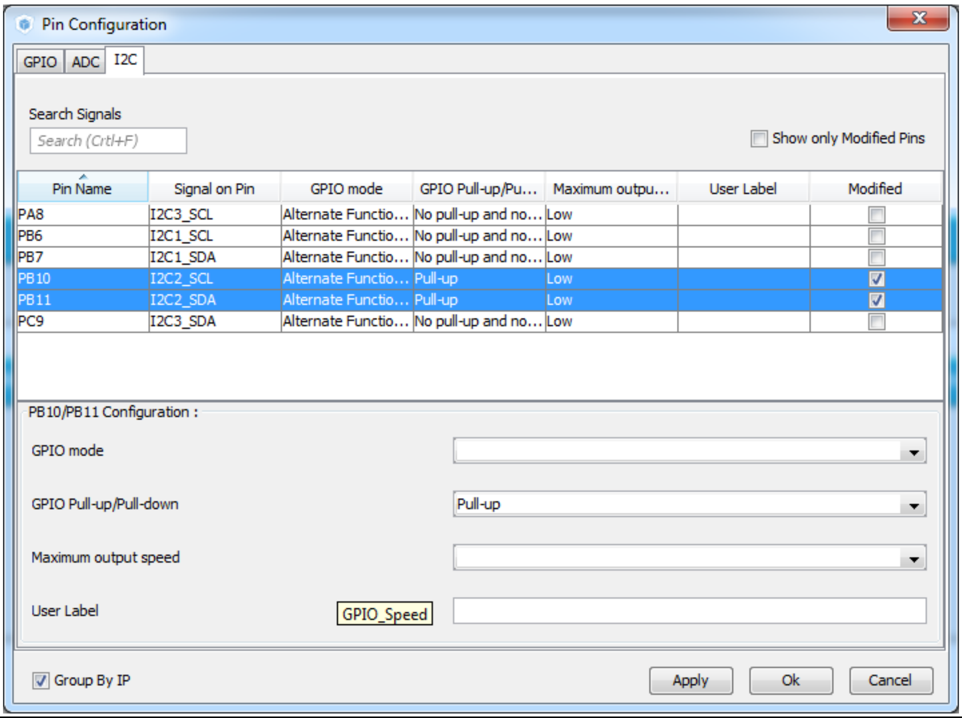

A **Group by IP** checkbox allows to group all instances of a peripheral under a same tab (see *Figure 44*).

**Figure 44. GPIO configuration grouped by IP**




As shown in *Figure 45*, row multi-selection can be performed to change a set of pins to a given configuration at the same time.

**Figure 45. Multiple Pins Configuration**



### 4.11.3 DMA Configuration window

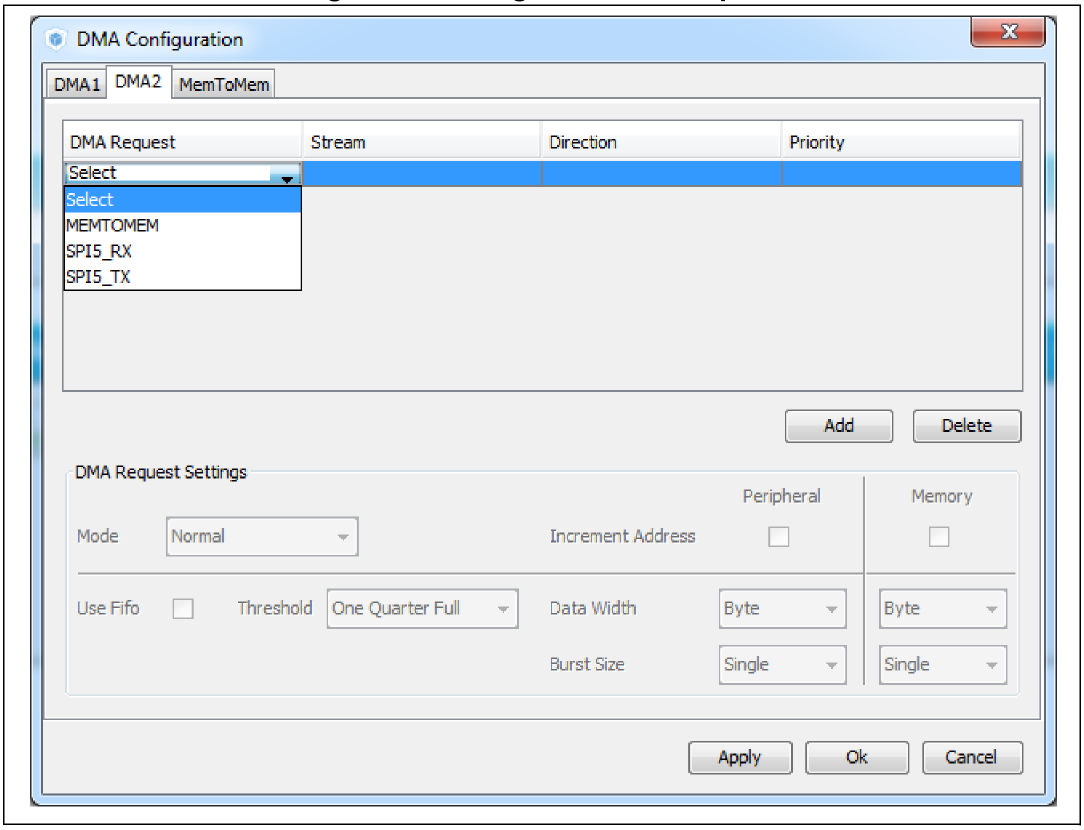
Click **DMA** in the **Configuration** pane to open the **DMA configuration** window.

This window allows to configure the generic DMA controllers available on the MCU. The DMA interfaces allow to perform data transfers between memories and peripherals while the CPU is running, and memory to memory transfers (if supported).

*Note: Some IPs such as **USB** or **Ethernet**, have their own DMA controller, which is enabled by default or via the IP configuration window.*

Clicking **Add** in the **DMA configuration** window adds a new line at the end of the DMA configuration table with a combo box proposing a choice of possible **DMA requests** to be mapped to peripherals signals (see *Figure 46*).

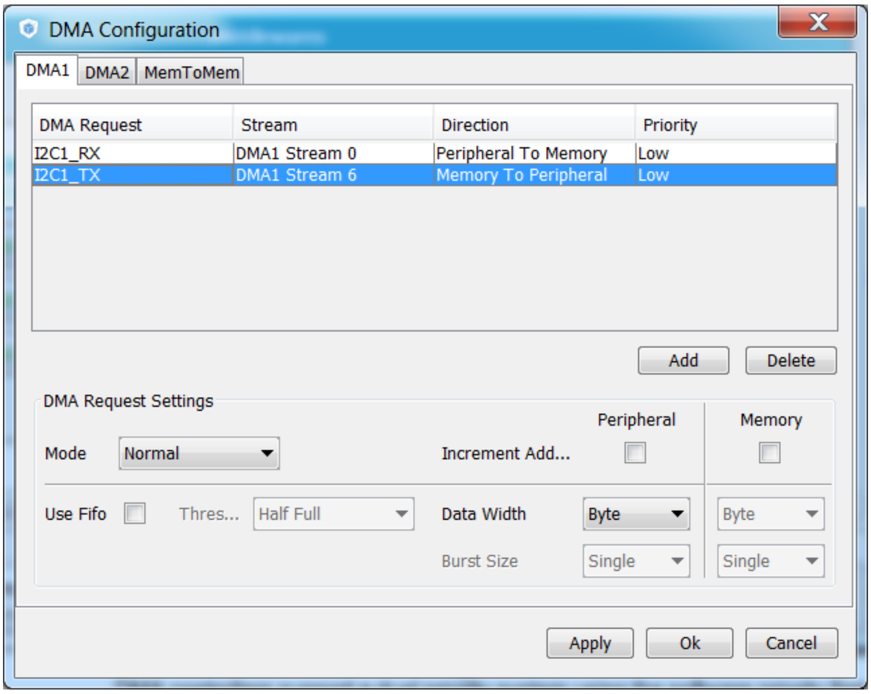
**Figure 46. Adding a new DMA request**




Selecting a DMA request automatically assigns a stream among all the streams available, a direction and a priority. The DMA request (called channel for STM32F4 MCUs) is used to reserve a stream to transfer data between peripherals and memories (see *Figure 47*). The stream priority will be used to decide which stream to select for the next DMA transfer.

DMA controllers support a dual priority system using the software priority first, and in case of equal software priorities, a hardware priority that is given by the stream number.

**Figure 47. DMA Configuration**

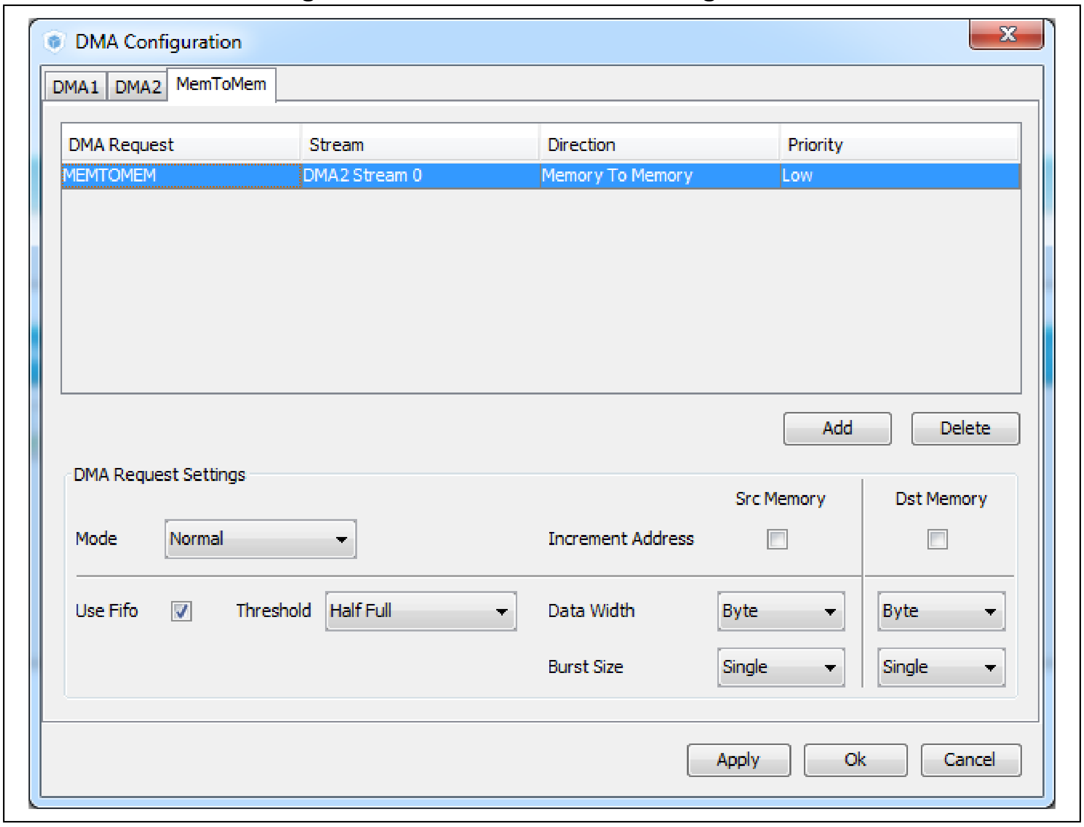



Additional DMA configuration settings can be done through the **DMA configuration** window:

- ***Mode***: regular mode, circular mode, or peripheral flow controller mode (only available for the SDIO IP).
- ***Increment Add:*** the type of peripheral address and memory address increment (fixed or post-incremented in which case the address is incremented after each transfer). Click the checkbox to enable the post-incremented mode.
- ***Peripheral data width***: 8, 16 or 32 bits
- Switching from the default direct mode to the *FIFO mode* with programmable *threshold*:
  a) Click the **Use FIFO** checkbox.
  b) Then, configure the ***peripheral and memory data width*** (8, 16 or 32 bits).
  c) Select between ***single transfer*** *and* ***burst transfer***. If you select burst transfer, choose a burst size (1, 4, 8 or 16).

In case of memory-to-memory transfer (MemtoMem), the DMA configuration applies to a source memory and a destination memory.

**Figure 48. DMA MemToMem configuration**




## 4.11.4 NVIC Configuration window

Click **NVIC** in the **Configuration** pane to open the Nested Vector interrupt controller configuration window (see *Figure 49*).

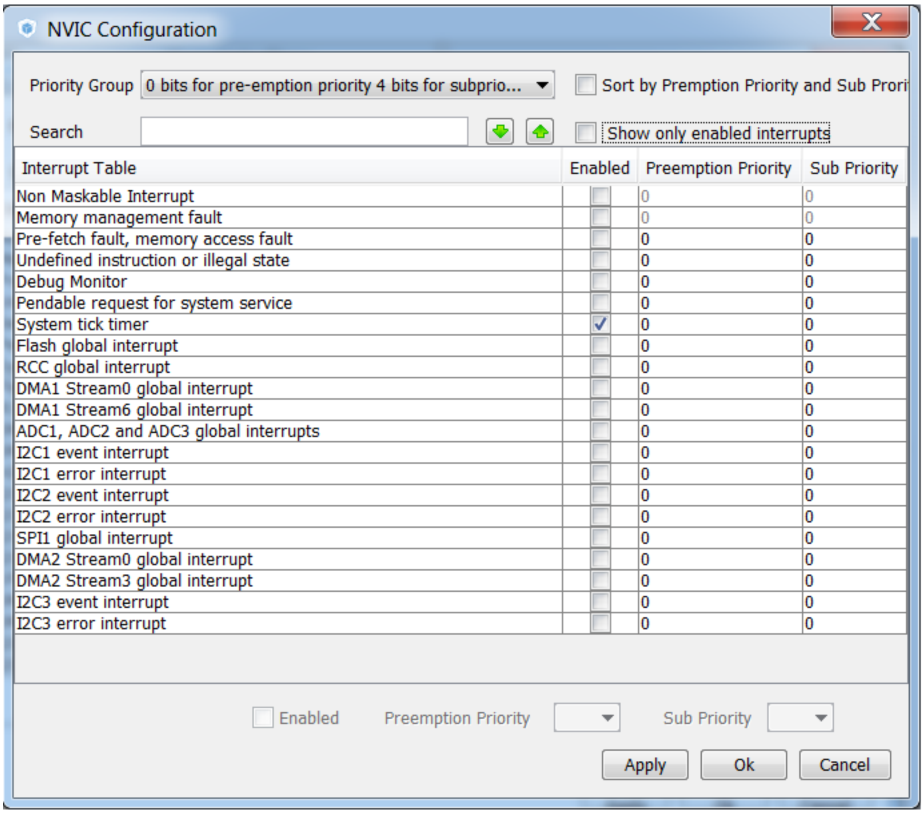
The NVIC window will not show all possible interrupts but only the ones available for the IPs selected in the **Pinout** and **Configuration** panes.

Check/Uncheck the **Show only enabled interrupts** box to filter or not on enabled interrupts.

Use the search field to filter out the interrupt vector table according to a string value. For example, after enabling UART IPs from the **Pinout** pane, type UART in the NVIC search field and click the green arrow next to it: all UART interrupts are displayed.

*Note: It is also possible to access IP dedicated interrupts using the NVIC tab in the IP configuration window.*

**Figure 49. NVIC Configuration window**




STM32CubeMX NVIC configuration consists in selecting a priority group, enabling/disabling interrupts and configuring interrupts priority levels (pre-emption and sub-priority levels):

1. Select a **priority group**

   Several bits allow to define NVIC priority levels. These bits are divided in two priority groups corresponding to two priority types: pre-emption priority and sub-priority. For example, in the case of STM32F4 MCUs, the NVIC priority group 0 corresponds to 0-bit pre-emption and 4-bit sub-priority.

2. In the interrupt table, click one or more rows to select one or more interrupt vectors. Use the widgets below the interrupt table to configure the vectors one by one or several at a time:
   - **Enable checkbox**: check/uncheck to enable/disable the interrupt.
   - **Pre-emption priority**: select a priority level. The pre-emption priority defines the ability of one interrupt to interrupt another.
   - **Sub-priority**: select a priority level. The sub-priority defines the interrupt priority level.
   - Click **Apply** to save changes, and **OK** to close the window.

## 4.12 Clock tree configuration view

The **Clock tree** view is available from the STM32CubeMX menu (see *Figure 17*). It helps configuring the microcontroller clocks according to the user application requirements.

STM32CubeMX generates the corresponding initialization code:

- main.c with relevant HAL_RCC structure initializations and function calls
- stm32xxxx_hal_conf.h for oscillator frequencies and $V_{DD}$ values.

### 4.12.1 Clock tree configuration functions

When using external clock sources, the user must previously enable them from the **Pinout** view available under the RCC peripheral.

Some other paths are grayed out. To become active, the peripheral must be properly configured in the **Pinout** view (e.g. USB). This view allows to:

- Enter a frequency value for the CPU Clock (HCLK), buses or peripheral clocks

  STM32CubeMX tries to propose a clock tree configuration that reaches the desired frequency while adjusting prescalers and dividers and taking into account other peripheral constraints (such as USB clock minimum value). If no solution can be found, STM32CubeMX proposes to switch to a different clock source or can even conclude that no solution matches the desired frequency.
- Lock the frequency fields for which the current value should be preserved.

  Right click a frequency field and select **Lock** to preserve the value currently assigned when STM32CubeMX will search for a new clock configuration solution.

  The user can unlock the locked frequency fields when the preservation is no longer necessary.
- Select the clock source that will drive the system clock (SYSCLK)
  - External oscillator clock (HSE) for a user defined frequency.
  - Internal oscillator clock (HSI) for the defined fixed frequency.
  - Main PLL clock
- Select secondary sources (as available for the product):
  - Low-speed internal (LSI) or external (LSE) clock
  - I2S input clock
  - …
- Select prescalers, dividers and multipliers values.
- Enable the Clock Security system (CSS) on HSE when it is supported by the MCU. This feature is available only when the HSE clock is used as the system clock source directly or indirectly through the PLL. It allows detecting HSE failure and inform the software about it, thus allowing the MCU to perform rescue operations.
- Enable the CSS on LSE when it is supported by the MCU.

  This feature is available only when the LSE and LSI are enabled and after the RTC or LCD clock sources have been selected to be either LSE or LSI.

*Note: To be available from the clock tree, external clocks, I2S input clock, and master clocks shall be enabled in RCC configuration in the* ***Pinout*** *view. This information is also available as tooltips.*

The tool will automatically perform the following operations:

- Adjust bus frequencies, timers, peripherals and master output clocks according to user selection of clock sources, clock frequencies and prescalers/multipliers/dividers values.
- Check the validity of user settings.
- Highlight invalid settings in red and provide tooltips to guide the user to achieve a valid configuration.

The Clock tree view is adjusted according to the RCC settings (configured in RCC IP pinout and configuration views) and vice versa:

- If in RCC **Pinout** view, the external and output clocks are enabled, they become configurable in the clock tree view.
- If in RCC Configuration view, the Timer prescaler is enabled, the choice of Timer clocks multipliers will be adjusted.

Conversely, the clock tree configuration may affect some RCC parameters in the configuration view:

- Flash latency: number of wait states automatically derived from $V_{DD}$ voltage, HCLK frequency, and power over-drive state.
- Power regulator voltage scale: automatically derived from HCLK frequency.
- Power over-drive is enabled automatically according to SYSCLK and HCLK values. When the power drive is enabled, the maximum possible frequency values for AHB and APB domains are increased. They are displayed in the Clock tree view.

The default optimal system settings that is used at startup are defined in the *system_stm32f4xx.c* file. This file is copied by STM32CubeMX from the STM32CubeF4 firmware package. The switch to user defined clock settings is done afterwards in the main function.

*Figure 50* gives an example of Clock tree configuration view for an STM32F429x MCU and *Table 11* describes the widgets that can be used to configure each clock.

**Figure 50. STM32F429xx Clock Tree configuration view**

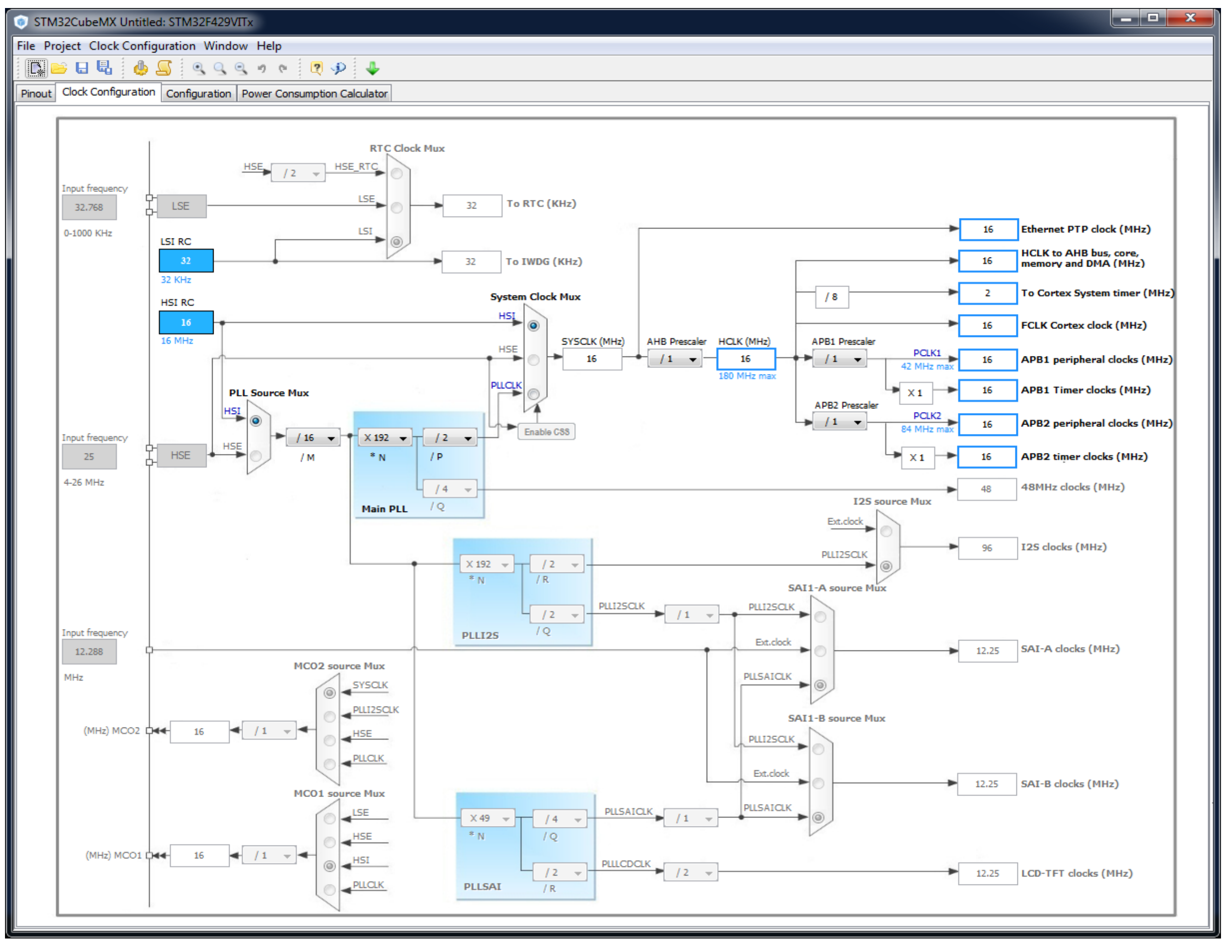



**Table 11. Clock tree view widget**

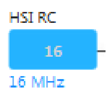
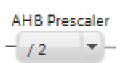
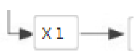
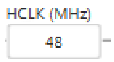
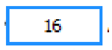
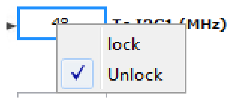
| Format | Configuration status of the Peripheral Instance |
|---|---|
|  | Active clock sources |
|  | Unavailable settings are blurred or grayed out (clock sources, dividers,…) |
|  | Gray drop down lists for prescalers, dividers, multipliers selection. |
|  | Multiplier selection |

**Table 11. Clock tree view widget (continued)**

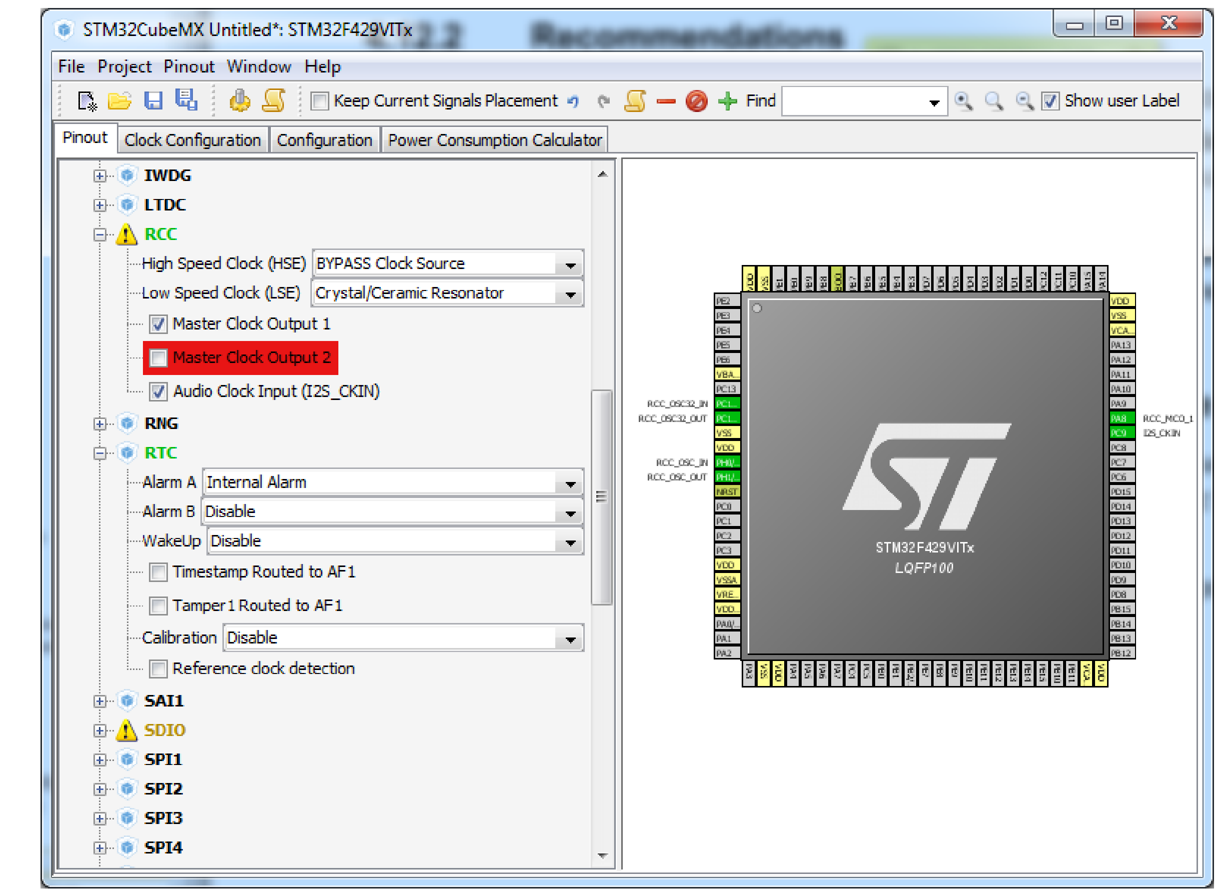
| Format | Configuration status of the Peripheral Instance |
|---|---|
|   | User defined frequency values |
|   | Automatically derived frequency values |
|   | User-modifiable frequency field |
|   | Right click blue border rectangles, to lock/unlock a frequency field. Lock to preserve the frequency value during clock tree configuration updates. |

### 4.12.2 Recommendations

The **Clock tree** view is not the only entry for clock configuration.

1. Go first through the **RCC IP pinout configuration** in the **Pinout** view to enable the clocks as needed: external clocks, master output clocks and Audio I2S input clock when available (see *Figure 51*).

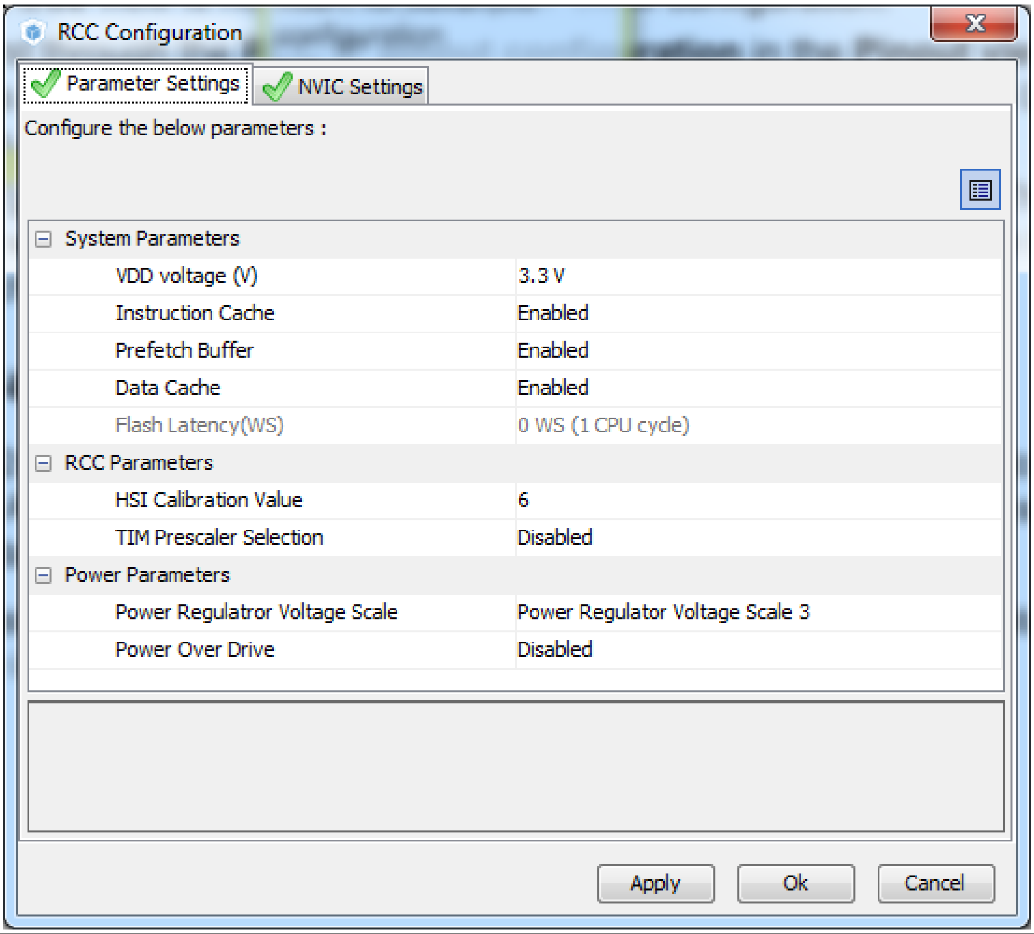
**Figure 51. Clock tree configuration: enabling RTC, RCC Clock source and outputs from Pinout view**




2. Then go to the **RCC IP configuration** in the **Configuration view**. The settings defined there for advanced configurations will be reflected in the **clock tree view.** The settings

defined in the clock tree view may change the settings in the RCC configuration (see *Figure 52*).

**Figure 52. Clock tree configuration: RCC Peripheral Advanced parameters**




### 4.12.3 STM32F43x/42x power-over drive feature

STM32F42x/43x MCUs implement a power over-drive feature allowing to work at the maximum AHB/APB bus frequencies (e.g., 180 MHz for HCLK) when a sufficient $V_{DD}$ supply voltage is applied (e.g $V_{DD}$ > 2.1 V).

*Table 12* lists the different parameters linked to the power over-drive feature and their availability in STM32CubeMX user interface.

**Table 12. Voltage scaling versus power over-drive and HCLK frequency**

| Parameter | STM32CubeMX panel | Value |
|---|---|---|
| $V_{DD}$ voltage | Configuration (RCC) | User-defined within a pre-defined range. Impacts power over-drive. |
| Power Regulator Voltage scaling | Configuration (RCC) | Automatically derived from HCLK frequency and power over-drive (see *Table 13*). |

**Table 12. Voltage scaling versus power over-drive and HCLK frequency**

| Parameter | STM32CubeMX panel | Value |
|---|---|---|
| Power Over Drive | Configuration (RCC) | This value is conditioned by HCLK and $V_{DD}$ value (see *Table 13*). It can be enabled only if $V_{DD} \geq 2.2$ V<br>When $V_{DD} \geq 2.2$ V, it is either automatically derived from HCLK or it can be configured by the user if multiple choices are possible (e.g., HCLK = 130 MHz) |
| HCLK/AHB clock maximum frequency value | Clock Configuration | Displayed in blue to indicate the maximum possible value. For example: maximum value is 168 MHz for HCLK when power over-drive cannot be activated (when $V_{DD} \leq 2.1$ V), otherwise it is 180 MHz. |
| APB1/APB2 clock maximum frequency value | Clock Configuration | Displayed in blue to indicate maximum possible value |

*Table 13* gives the relations between power-over drive mode and HCLK frequency.

**Table 13. Relations between power over-drive and HCLK frequency**

| HCLK frequency range: $V_{DD}$ > 2.1 V required to enable power over-drive (POD) | Corresponding voltage scaling and power over-drive (POD) |
|---|---|
| ≤120 MHz | Scale 3<br>POD is disabled |
| 120 to 14 MHz | Scale 2<br>POD can be either disabled or enabled |
| 144 to 168 MHz | Scale 1 when POD is disabled<br>Scale 2 when POD is enabled |
| 168 to 180 MHz | POD must be enabled<br>Scale 1 (otherwise frequency range not supported) |

### 4.12.4 Clock tree glossary

**Table 14. Glossary**

| Acronym | Definition |
|---|---|
| HSI | High Speed Internal oscillator: enabled after reset, lower accuracy than HSE. |
| HSE | High Speed External oscillator: requires an external clock circuit. |
| PLL | Phase Locked Loop: used to multiply above clock sources. |
| LSI | Low Speed Internal clock: low power clocks usually used for watchdog timers. |

**Table 14. Glossary (continued)**

| Acronym | Definition |
|---|---|
| LSE | Low Speed External clock: powered by an external clock. |
| SYSCLK | System clock |
| HCLK | Internal AHB clock frequency |
| FCLK | Cortex free running clock |
| AHB | Advanced High Performance Bus |
| APB1 | Low speed Advanced Peripheral Bus |
| APB2 | High speed Advanced Peripheral Bus |

## 4.13 Power Consumption Calculator (PCC) view

Select the PCC tab from STM32CubeMX main window to display the PCC view (see *Figure 17*). Given a microcontroller, a battery model and a user-defined power sequence, STM32CubeMX provides an estimation of the following parameters:

- Average power consumption
- Battery life
- Average DMIPS.

Power consumption and DMIPS data are directly taken from the MCU datasheet and are neither interpolated nor extrapolated.

STM32CubeMX supports user-defined batteries through an interface to add and remove batteries.

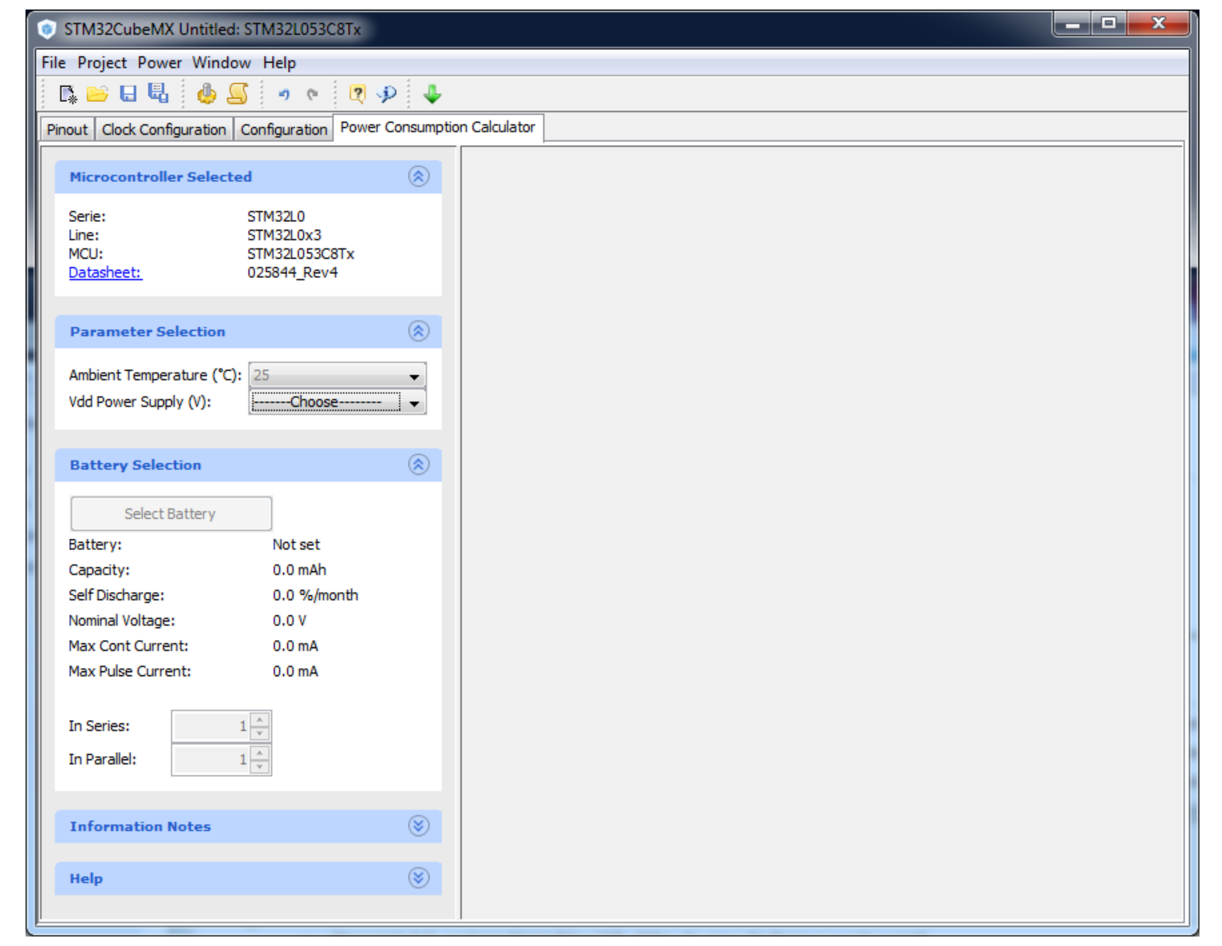
For each step, the user can choose VBUS as possible power source instead of the battery. This will impact the battery life estimation. If power consumption measurements are available at different voltage levels, STM32CubeMX will also propose a choice of voltage values (see *Figure 59*).

An additional option, the transition checker, is available for STM32L0 and STM32L1 series. When it is enabled, the transition checker detects invalid transitions within the currently configured sequence. It ensures that only possible transitions are proposed to the user when a new step is added.

### 4.13.1 Building a power consumption sequence

The default starting view is shown in *Figure 53*.

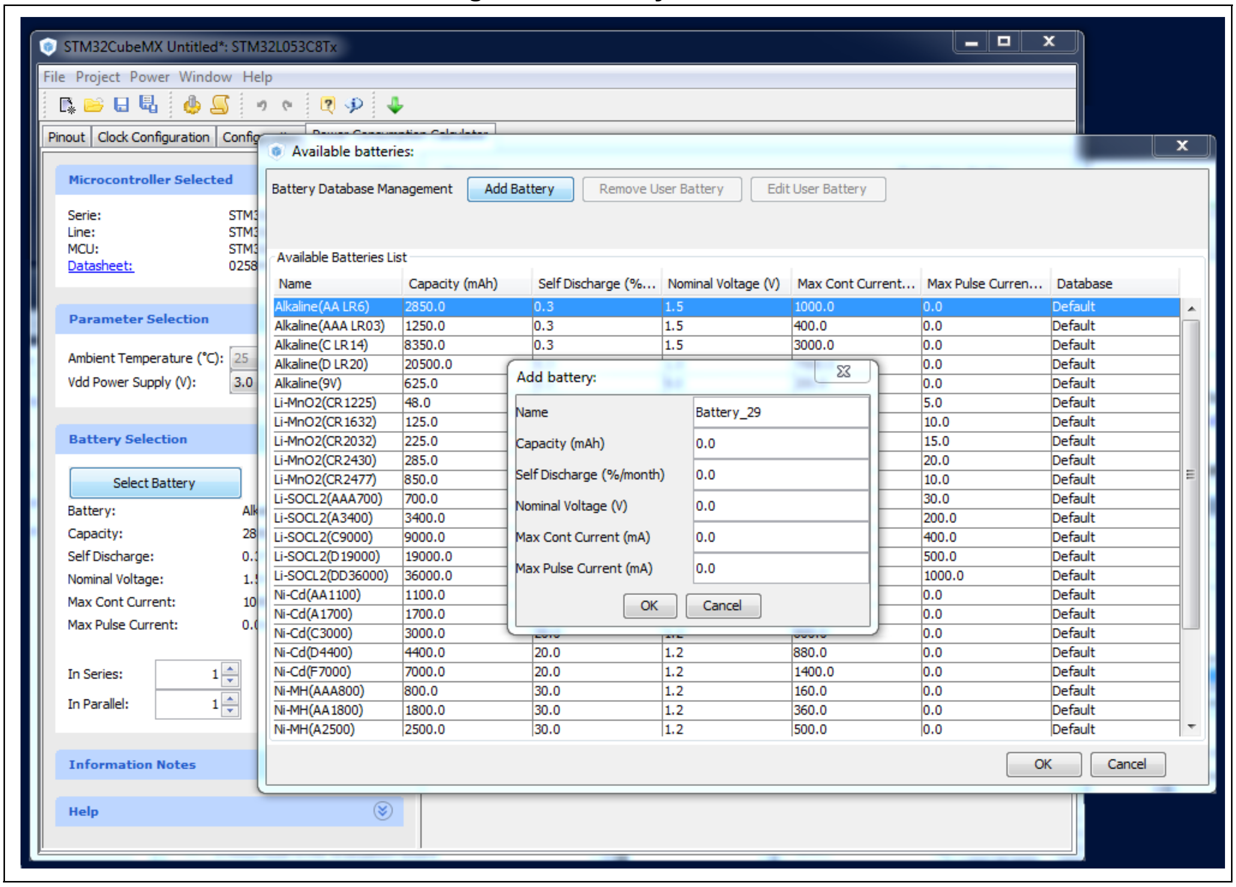
**Figure 53. Power consumption calculator default view**




From this view, the user must select a $V_{DD}$ **value** (when multiple choice are available) and a **battery model** (optional).

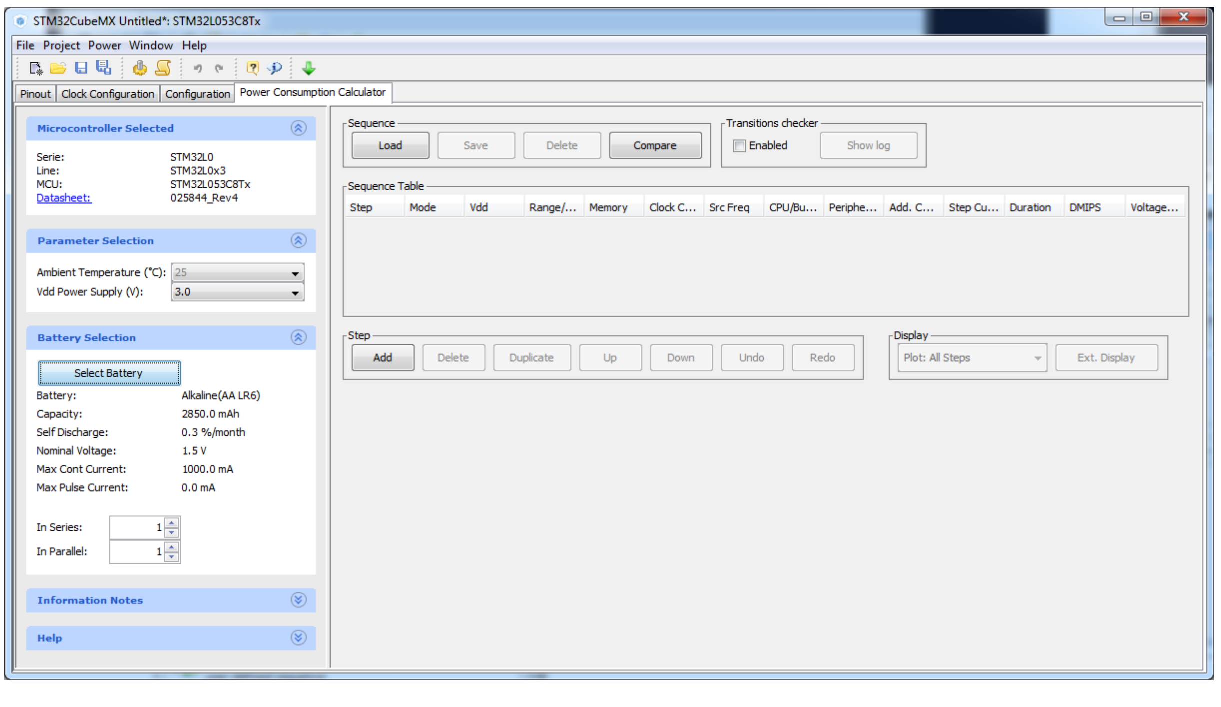
The user can select a pre-defined battery or choose to specify a new battery that best matches his application (see *Figure 54*).

**Figure 54. Battery selection**



The user can now proceed and build a power sequence by clicking the **Add** Step button to add a step in the sequence (see *Figure 55* and *Figure 59*). Note that once a step is configured, its resulting consumption is provided in the window.

**Figure 55. Building a power consumption sequence**

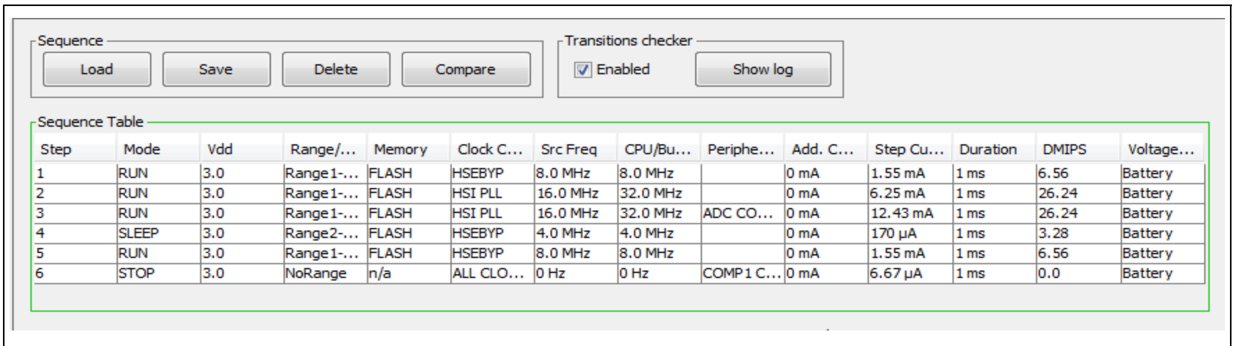



Enabling the transition checker option prior to sequence configuration ensures the user will be able to select only valid transition steps.

Enabling the transition checker option on an already configured sequence will highlight the sequence in green (green frame) if all transitions are valid (see *Figure 56*), or in red if at least one transition is invalid (red frame with description of invalid step highlighted in red) (see *Figure 57*).

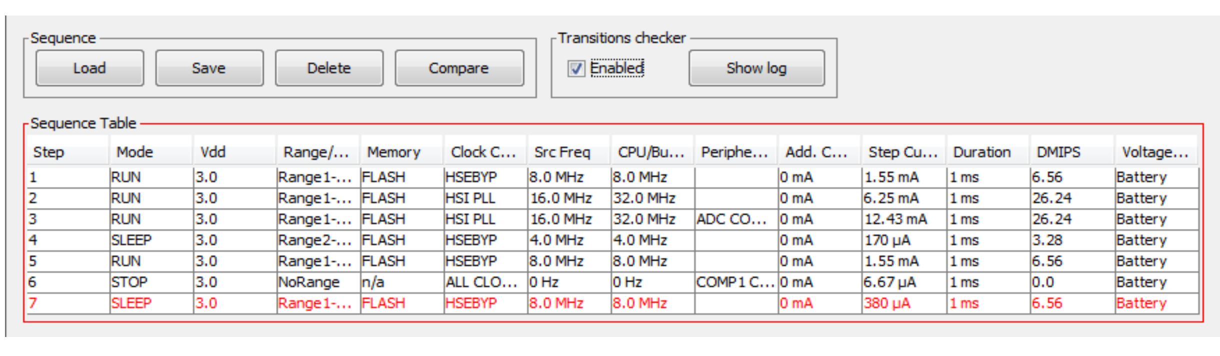
In this case, the user can click the **Show log** button to find out how to solve the transition issue (see *Figure 58*).

**Figure 56. Enabling the transition checker option on an already configured sequence - all transitions valid**




| Step | Mode | Vdd | Range/... | Memory | Clock C... | Src Freq | CPU/Bu... | Periphe... | Add. C... | Step Cu... | Duration | DMIPS | Voltage... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | RUN | 3.0 | Range1-... | FLASH | HSEBYP | 8.0 MHz | 8.0 MHz | | 0 mA | 1.55 mA | 1 ms | 6.56 | Battery |
| 2 | RUN | 3.0 | Range1-... | FLASH | HSI PLL | 16.0 MHz | 32.0 MHz | | 0 mA | 6.25 mA | 1 ms | 26.24 | Battery |
| 3 | RUN | 3.0 | Range1-... | FLASH | HSI PLL | 16.0 MHz | 32.0 MHz | ADC CO... | 0 mA | 12.43 mA | 1 ms | 26.24 | Battery |
| 4 | SLEEP | 3.0 | Range2-... | FLASH | HSEBYP | 4.0 MHz | 4.0 MHz | | 0 mA | 170 µA | 1 ms | 3.28 | Battery |
| 5 | RUN | 3.0 | Range1-... | FLASH | HSEBYP | 8.0 MHz | 8.0 MHz | | 0 mA | 1.55 mA | 1 ms | 6.56 | Battery |
| 6 | STOP | 3.0 | NoRange | n/a | ALL CLO... | 0 Hz | 0 Hz | COMP1 C... | 0 mA | 6.67 µA | 1 ms | 0.0 | Battery |

**Figure 57. Enabling the transition checker option on an already configured sequence - at least one transition invalid**




| Step | Mode | Vdd | Range/... | Memory | Clock C... | Src Freq | CPU/Bu... | Periphe... | Add. C... | Step Cu... | Duration | DMIPS | Voltage... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | RUN | 3.0 | Range1-... | FLASH | HSEBYP | 8.0 MHz | 8.0 MHz | | 0 mA | 1.55 mA | 1 ms | 6.56 | Battery |
| 2 | RUN | 3.0 | Range1-... | FLASH | HSI PLL | 16.0 MHz | 32.0 MHz | | 0 mA | 6.25 mA | 1 ms | 26.24 | Battery |
| 3 | RUN | 3.0 | Range1-... | FLASH | HSI PLL | 16.0 MHz | 32.0 MHz | ADC CO... | 0 mA | 12.43 mA | 1 ms | 26.24 | Battery |
| 4 | SLEEP | 3.0 | Range2-... | FLASH | HSEBYP | 4.0 MHz | 4.0 MHz | | 0 mA | 170 µA | 1 ms | 3.28 | Battery |
| 5 | RUN | 3.0 | Range1-... | FLASH | HSEBYP | 8.0 MHz | 8.0 MHz | | 0 mA | 1.55 mA | 1 ms | 6.56 | Battery |
| 6 | STOP | 3.0 | NoRange | n/a | ALL CLO... | 0 Hz | 0 Hz | COMP1 C... | 0 mA | 6.67 µA | 1 ms | 0.0 | Battery |
| 7 | SLEEP | 3.0 | Range1-... | FLASH | HSEBYP | 8.0 MHz | 8.0 MHz | | 0 mA | 380 µA | 1 ms | 6.56 | Battery |

**Figure 58. Transition checker option -show log**

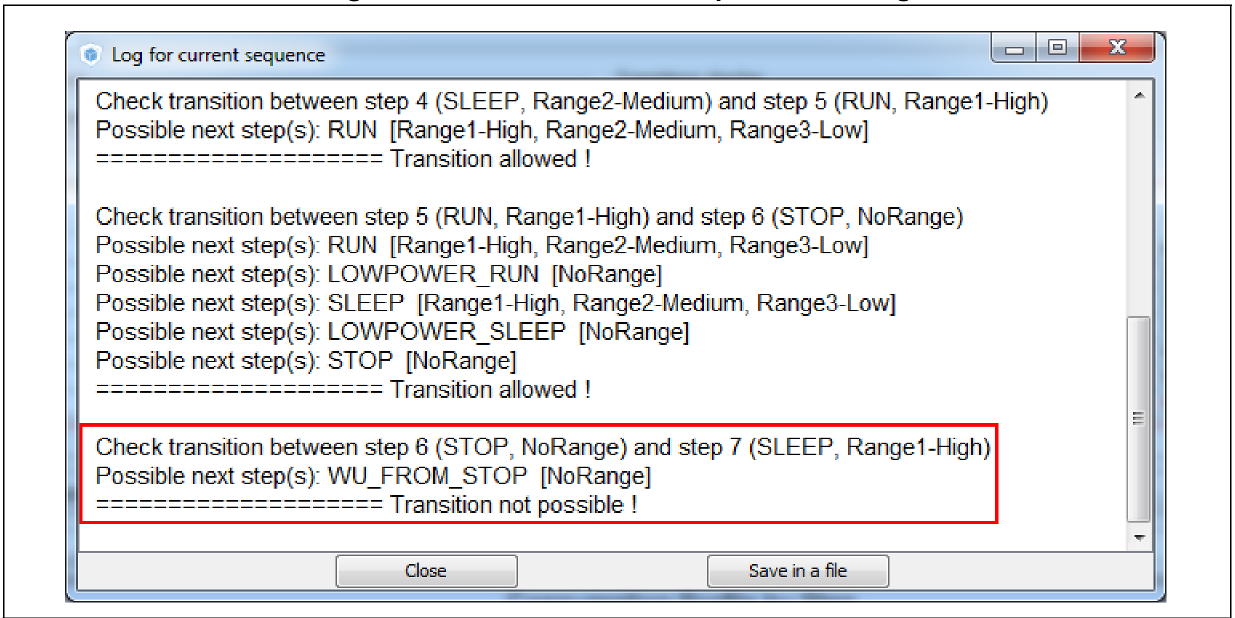
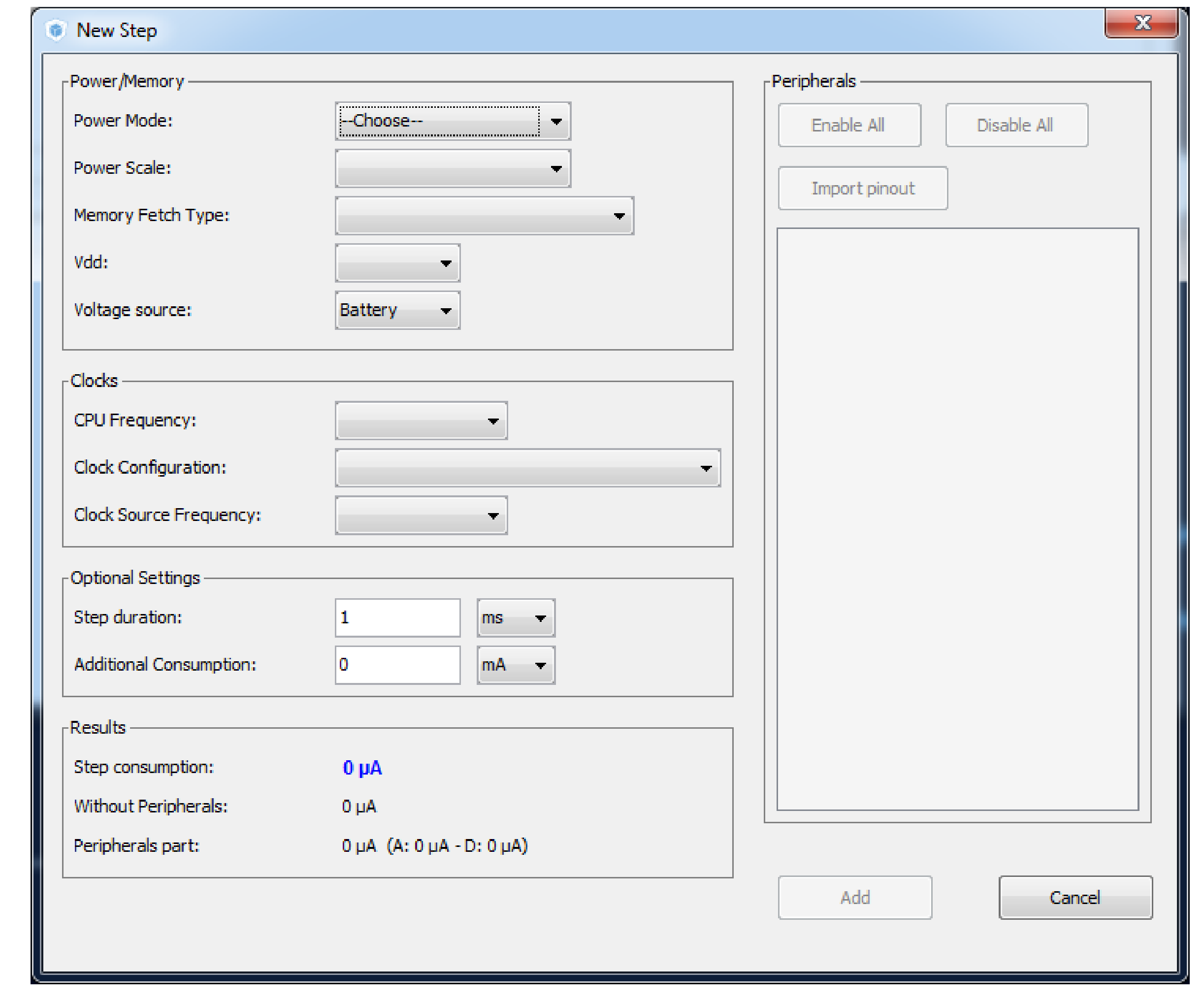


**Figure 59. Power consumption sequence: new step default view (STM32F4 example)**

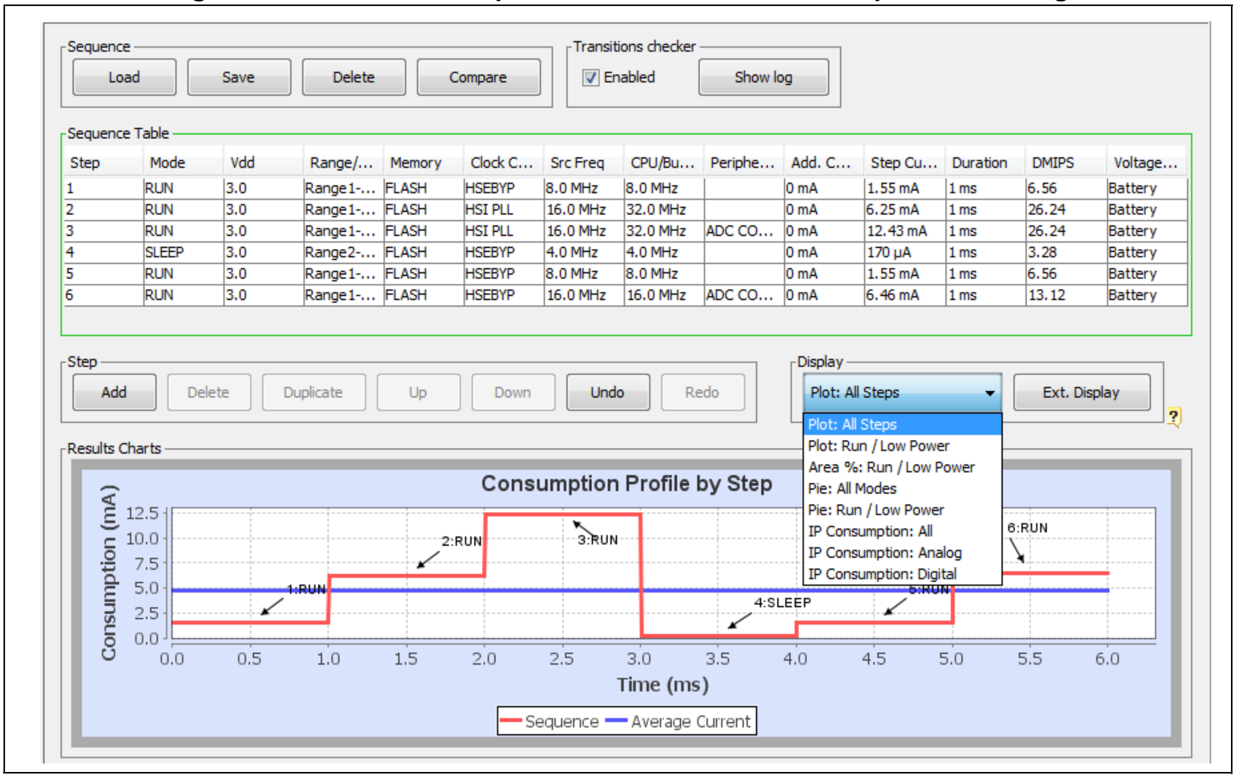


### 4.13.2 User-defined power sequence and results

The configuration of a power sequence leads to an update of the PCC view (see *Figure 60*):

- The sequence table shows all steps and step parameters values.
- The sequence chart area shows different views of the power sequence according to a display type (e.g. plot all steps, plot low power versus run modes, ..)
- The results summary provides the total sequence time, estimate of the average power consumption, DMIPS, and battery lifetime provided a valid battery configuration has been selected.

**Figure 60. Power Consumption Calculator view after sequence building**

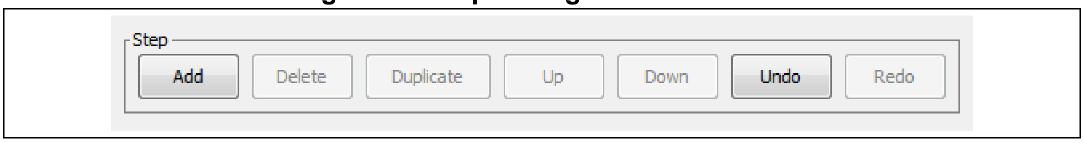



### Managing sequence steps

Steps can be reorganized within a sequence (**Add** new, **Delete** a step, **Duplicate** a step, move **Up** or **Down** in the sequence) using the set of Step buttons (see *Figure 61*).

The user can undo or redo the last configuration actions by clicking the **Undo** button in the PCC view or the Undo icon from the main toolbar

**Figure 61. Step management functions**



### Managing the whole sequence

The current sequence can be saved or deleted. In addition, a previously saved sequence can be either loaded in the current view or opened for comparison (see *Figure 62*).

**Figure 62. Sequence table management functions**


To load a previously saved sequence:

1. Click the load button.
2. Browse to select the sequence to load.

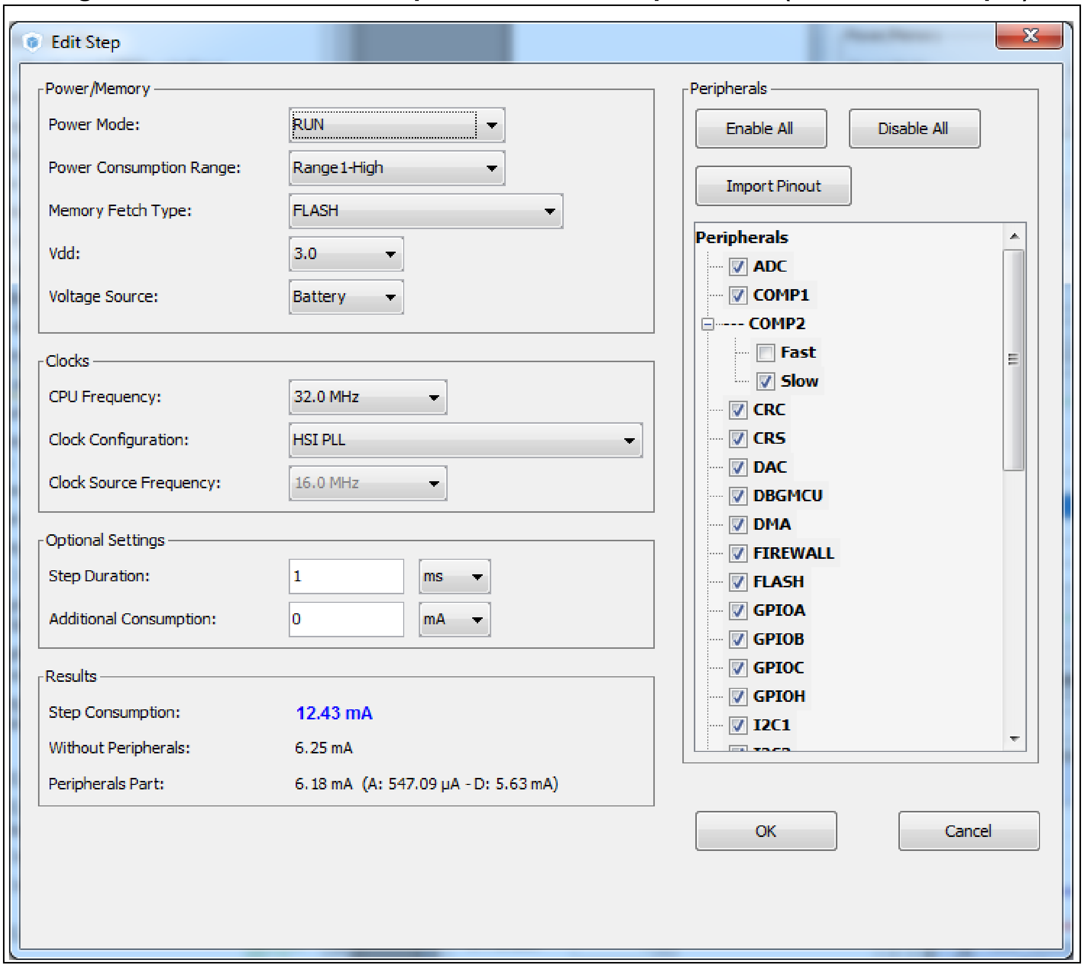
To open a previously saved sequence for comparison:

1. Click the **Compare** button.
2. Browse and select the .pcs sequence file to be compared with the current sequence. A new window opens showing the selected sequence details.

### Editing sequence steps

To edit a step, double-click it in the sequence table. This opens the **Edit Step** window as shown below.

**Figure 63. STM32L0 PCC step edited in Edit Step window (STM32L0 example)**




### Configuring sequence steps

Several parameters must be configured. Their naming may differ according to the MCU series selected. For details on each parameter, refer to *Section 4.13.3: Power sequence step parameters glossary* and to *Appendix D: STM32 microcontrollers power consumption parameters* or refer to the electrical characteristics section of the MCU datasheet.

Figure 64. Power consumption sequence: new step configured (STM32F4 example)

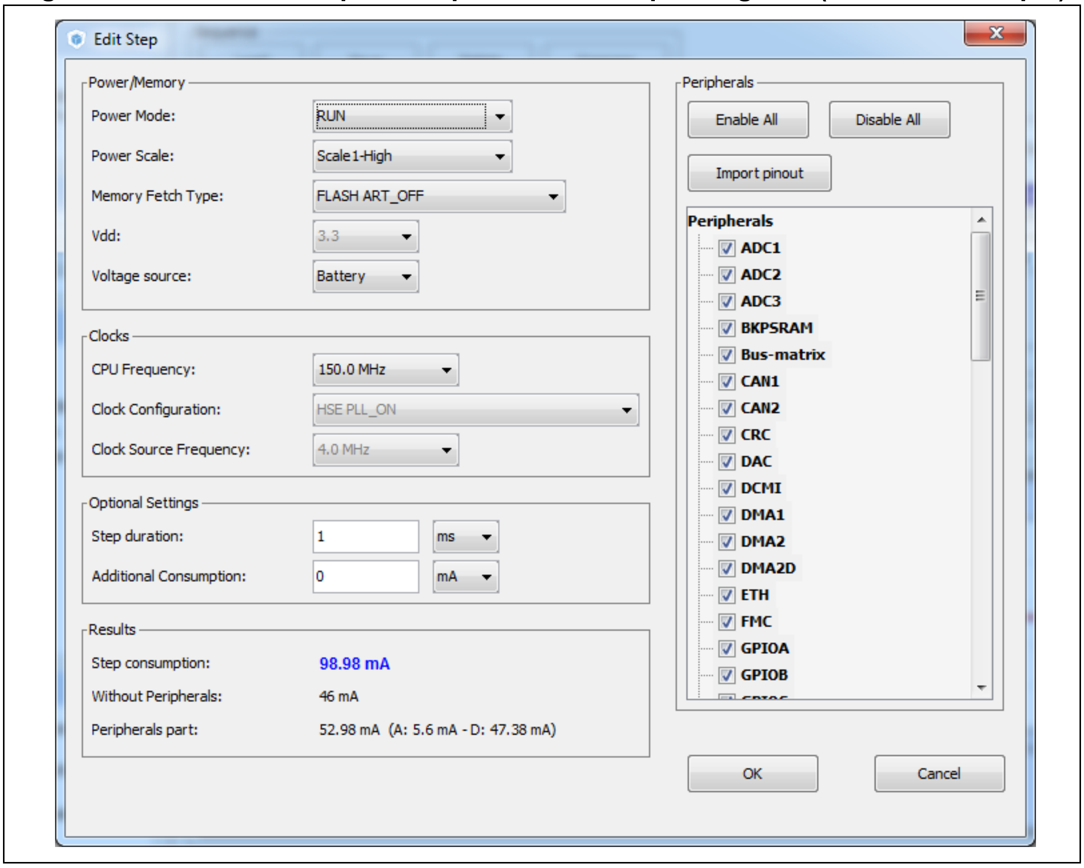



*Figure 65* illustrates the example of the ADC configuration in the **Pinout** view: clicking **Import Pinout** in the PCC view selects the ADC IP and GPIO A (*Figure 66*). The **Import pinout** button allows to automatically select the IPs that have been configured in the **Pinout** view.

**Figure 65. ADC selected in Pinout view**

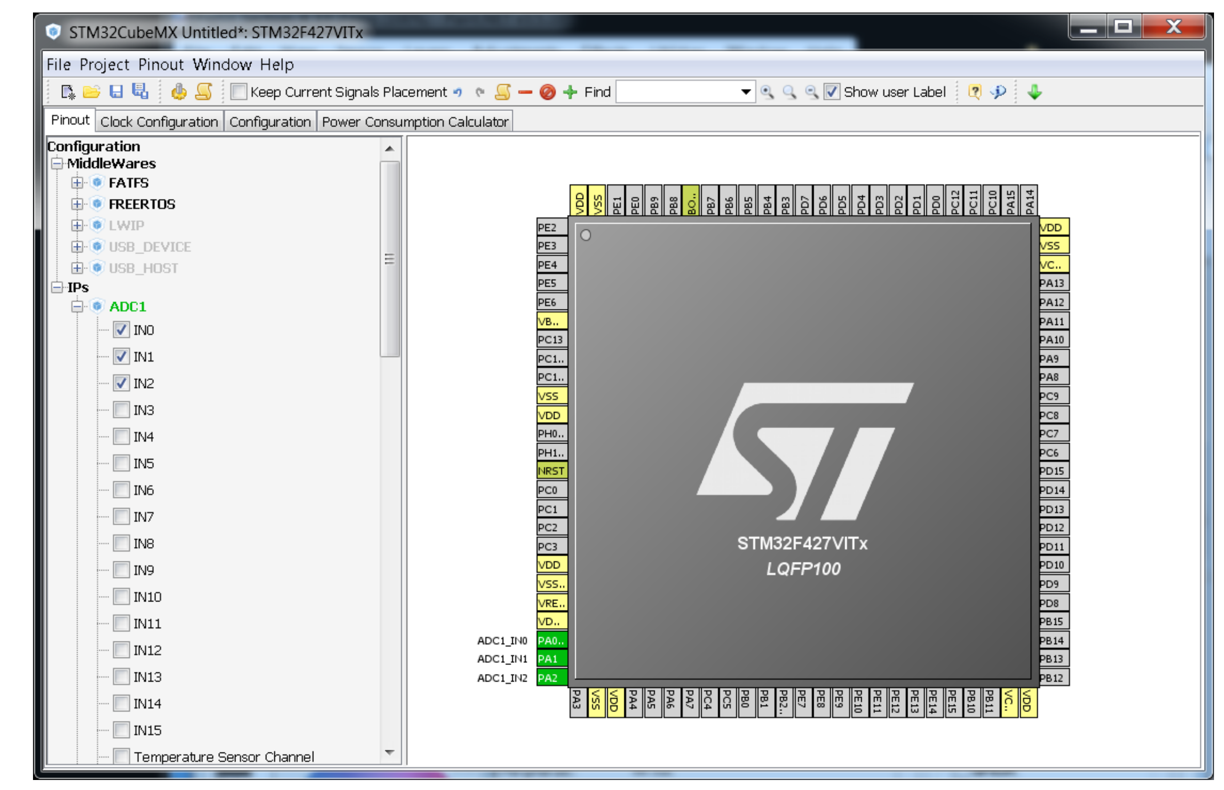
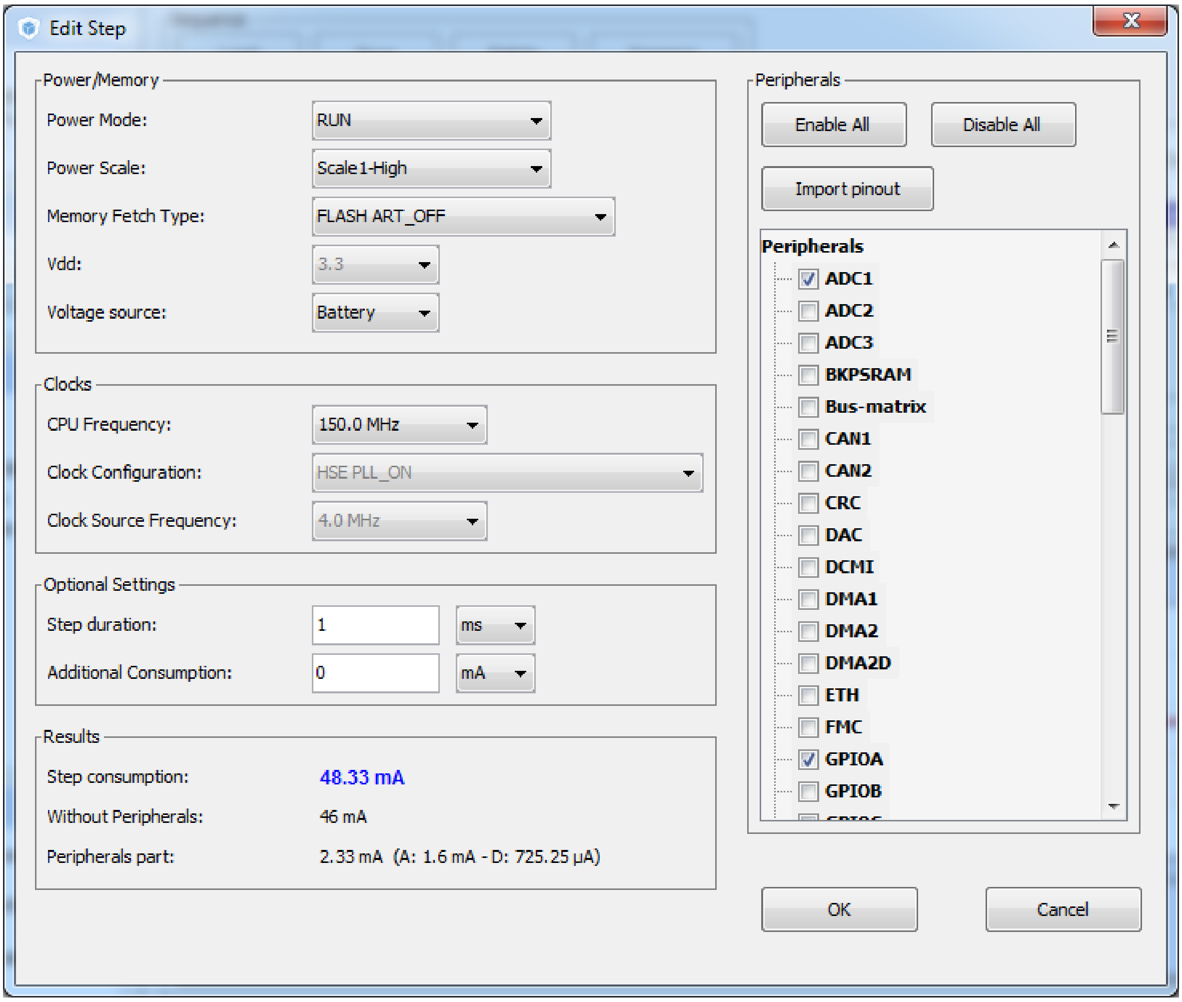


**Figure 66. PCC Step configuration window: ADC enabled using import pinout**

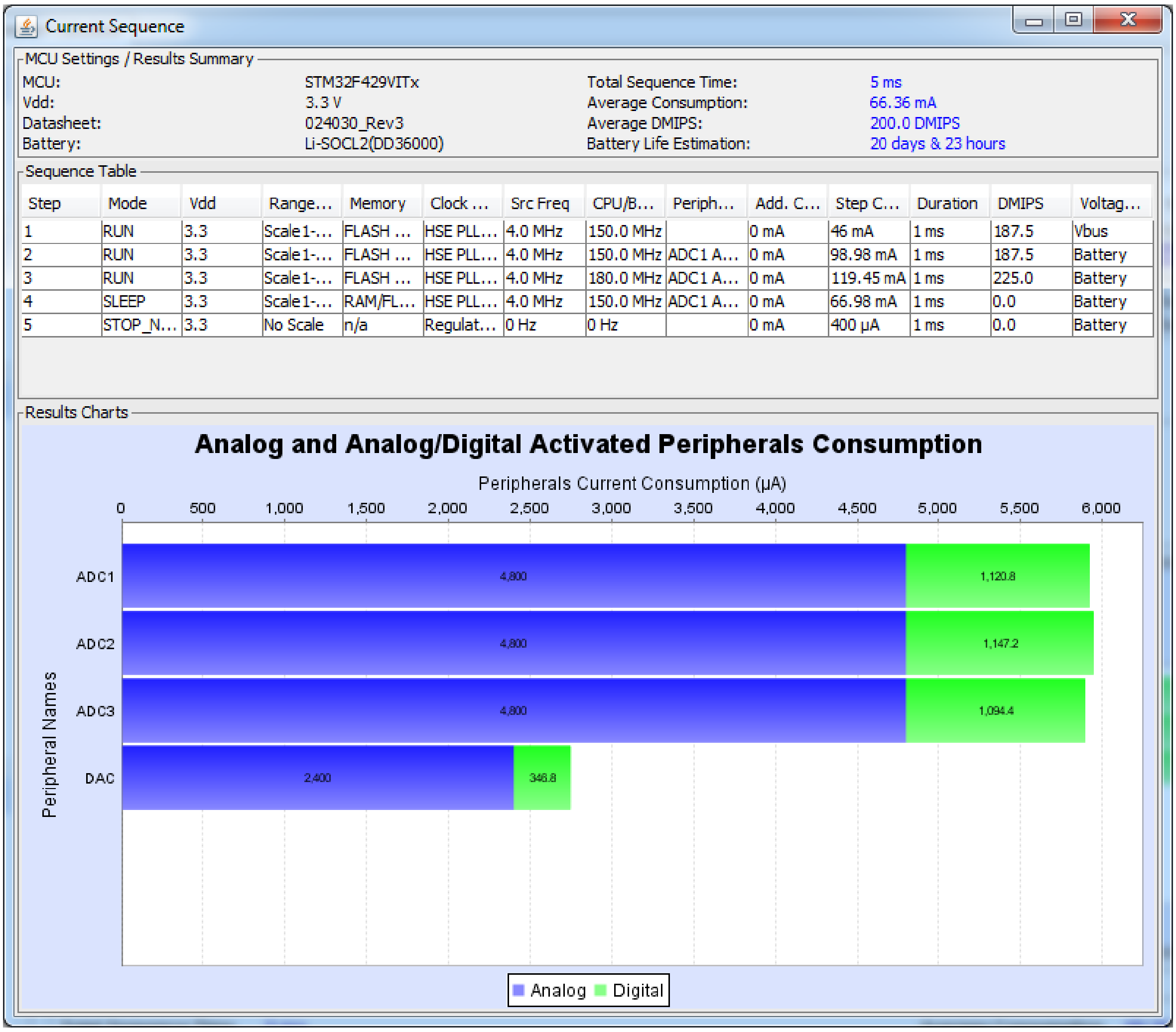



### Managing the results charts and display options

In the Display section, select the type of chart to display (sequence steps, pie charts, consumption per IPs, ...). You can also click **External Display** to open the charts in dedicated windows (see *Figure 67*).

Right-click on the chart to access the contextual menus: **Properties**, **Copy**, **Save** as png picture file, **Print**, **Zoom** menus, and **Auto Range** to reset to the original view before zoom operations. **Zooming** can also be achieved by mouse selecting from left to right a zone in the chart and **Zoom reset** by clicking the chart and dragging the mouse to the left.

**Figure 67. Power Consumption: Peripherals Consumption Chart**

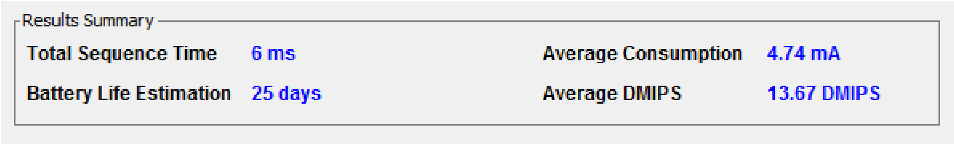



### Overview of the Results summary section

This section provides the following information (see *Figure 68*):

- Total sequence time as the sum of the sequence steps durations.
- Average consumption as the sum of each step consumption weighed by the step duration.
- The average DMIPS (Dhrystone Million Instructions per Second) based on Dhrystone benchmark, highlighting the CPU performance for the defined sequence.
- Battery life estimation for the selected battery model, based on the average power consumption and the battery self-discharge.

**Figure 68. Description of the result section**




### 4.13.3 Power sequence step parameters glossary

The parameters that characterize power sequence steps are the following (refer to *Appendix D: STM32 microcontrollers power consumption parameters* for more details):

- Power modes

  To save energy, it is recommended to switch the microcontroller operating mode from running mode, where a maximum power is required, to a low-power mode requiring limited resources.

- $V_{CORE}$ range (STM32L1) or Power scale (STM32F4)

  These parameters are set by software to control the power supply range for digital peripherals.

- Memory Fetch Type

  This field proposes the possible memory locations for application C code execution. It can be either RAM, FLASH or FLASH with ART ON or OFF (only for families that feature a proprietary Adaptive real-time (ART) memory accelerator which increases the program execution speed when executing from Flash memory).

  The performance achieved thanks to the ART accelerator is equivalent to 0 wait state program execution from Flash memory. In terms of power consumption, it is equivalent to program execution from RAM. In addition, STM32CubeMX uses the same selection choice to cover both settings, RAM and Flash with ART ON.

- Clock Configuration

  This operation sets the AHB bus frequency or the CPU frequency that will be used for computing the microcontroller power consumption. When there is only one possible choice, the frequencies are automatically configured.

  The clock configuration drop-down list allows to configure the application clocks:

  – The internal or external oscillator sources: MSI, HSI, LSI, HSE or LSE),
  – The oscillator frequency,
  – Other determining parameters: PLL ON, LSE Bypass, AHB prescaler value, LCD with duty...

- Peripherals

  The peripheral list shows the peripherals available for the selected power mode. The power consumption is given assuming that peripherals are only clocked (e.g. not in use by a running program). Each peripheral can be enabled or disabled. Peripherals individual power consumptions are displayed in a tooltip. An overall consumption due to peripheral analog and digital parts is provided in the step Results section (see

*Figure 69*).

The user can select the peripherals relevant for the application:

– None (**Disable All**),

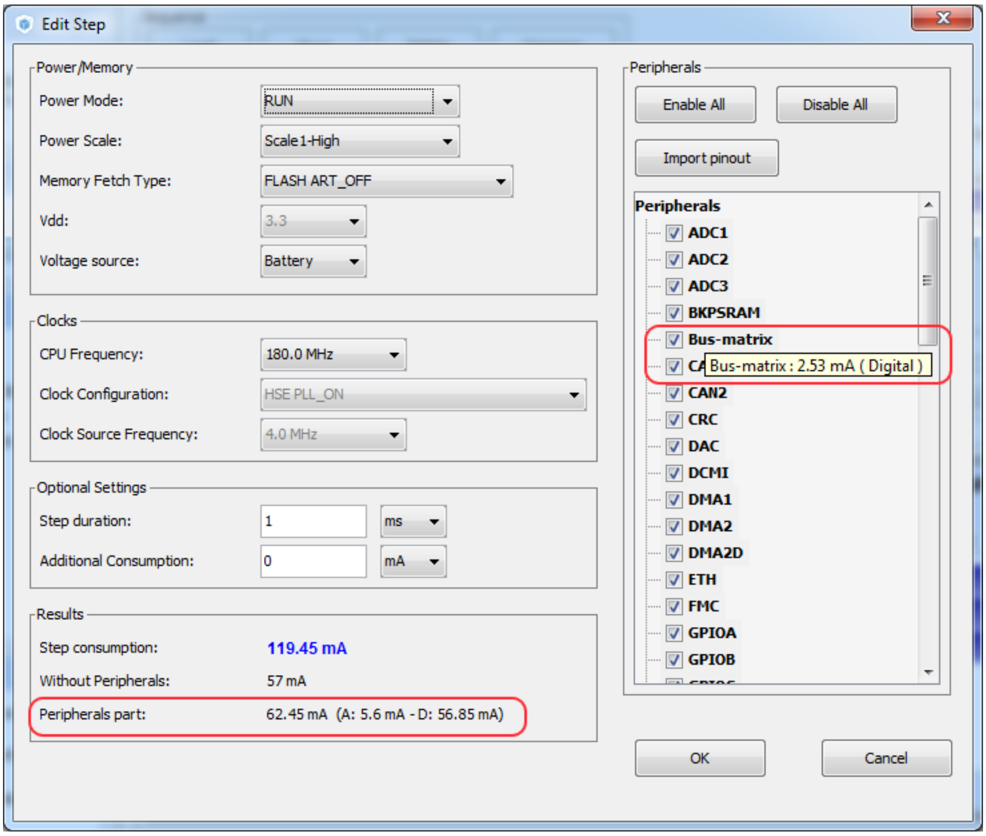
– Some (using IP individual checkbox),

– All (**Activate All**),

– Or all from the previously defined pinout configuration (**Import Pinout**).

Only the selected and enabled peripherals are taken into account when computing the power consumption.

**Figure 69. Peripheral power consumption tooltip**




- Step duration

  The user can change the default step duration value. When building a sequence, the user can either create steps according to the application actual power sequence or define them as a percentage spent in each mode. For example, if an application spends 30% in Run mode, 20% in Sleep and 50% in Stop, the user must configure a 3-step sequence consisting in 30 ms in Run, 20 ms in Sleep and 50 ms in Stop.

- Additional Consumption

  This field allows entering an additional consumption resulting from specific user configuration (e.g. MCU providing power supply to other connected devices).

### 4.13.4 Battery glossary

- Capacity (mAh)

  Amount of energy that can be delivered in a single battery discharge.
- Self-discharge (%/month)

  This percentage, over a specified period, represents the loss of battery capacity when the battery is not used (open-circuit conditions), as a result of internal leakage.
- Nominal voltage (V)

  Voltage supplied by a fully charged battery.
- Max. Continuous Current (mA)

  This current corresponds to the maximum current that can be delivered during the battery lifetime period without damaging the battery.
- Max. Pulse Current (mA)

  This is the maximum pulse current that can be delivered exceptionally, for instance when the application is switched on during the starting phase.

# 5 STM32CubeMX C Code generation overview

Refer to *Section 4.4.2: Project menu* for code generation and C project settings related topics.

## 5.1 Standard STM32Cube code generation

During the C code generation process, STM32CubeMX performs the following actions:

- If it is missing, it downloads the relevant STM32Cube firmware package from the user repository. STM32CubeMX repository folder is specified in the **Help > Updater settings** menu.
- It copies from the firmware package, the relevant files in *Drivers/CMSIS* and *Drivers/STM32F4_HAL_Driver* folders and in the *Middleware* folder if a middleware was selected.
- It generates the initialization C code ( .c/.h files) corresponding to the user MCU configuration and stores it in the *Inc* and *Src* folders. By default, the following files are included:
  - **stm32f4xx_hal_conf.h** file: this file defines the enabled HAL modules and sets some parameters (e.g. External High Speed oscillator frequency) to pre-defined default values or according to user configuration (clock tree).
  - **stm32f4xx_hal_msp.c** (MSP = MCU Support package): this file defines all initialization functions to configure the IP instances according to the user configuration (pin allocation, enabling of clock, use of DMA and Interrupts).
  - **main.c** is in charge of:

    Resetting the MCU to a known state by calling the *HAL_init()* function that resets all peripherals, initializes the Flash memory interface and the SysTick.

    Configuring and initializing the system clock.

    Configuring and initializing the GPIOs that are not used by IPs.

    Defining and calling, for each configured IP, an IP initialization function that defines a handle structure that will be passed to the corresponding IP *HAL init* function which in turn will call the IP HAL MSP initialization function. Note that when LwIP (respectively USB) middleware is used, the initialization C code for the underlying Ethernet (respectively USB IP) is moved from main.c to LwIP (respectively USB) initialization C code itself.
- It generates a *Projects* folder that contains the toolchain specific files that match the user project settings. Double-clicking the IDE specific project file launches the IDE and loads the project ready to be edited, built and debugged.

## 5.2 Custom code generation

STM32CubeMX supports custom code generation by means of a FreeMarker template engine (see http://www.freemarker.org): 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 70: 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")]
```

To generate a custom code, the user must place the Freemarker template file under STM32CubeMX installation path within the db/extra_templates folder (see *Figure 71: 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 72: 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 73: User custom folder for templates*).

**Figure 70. extra_templates folder – default content**

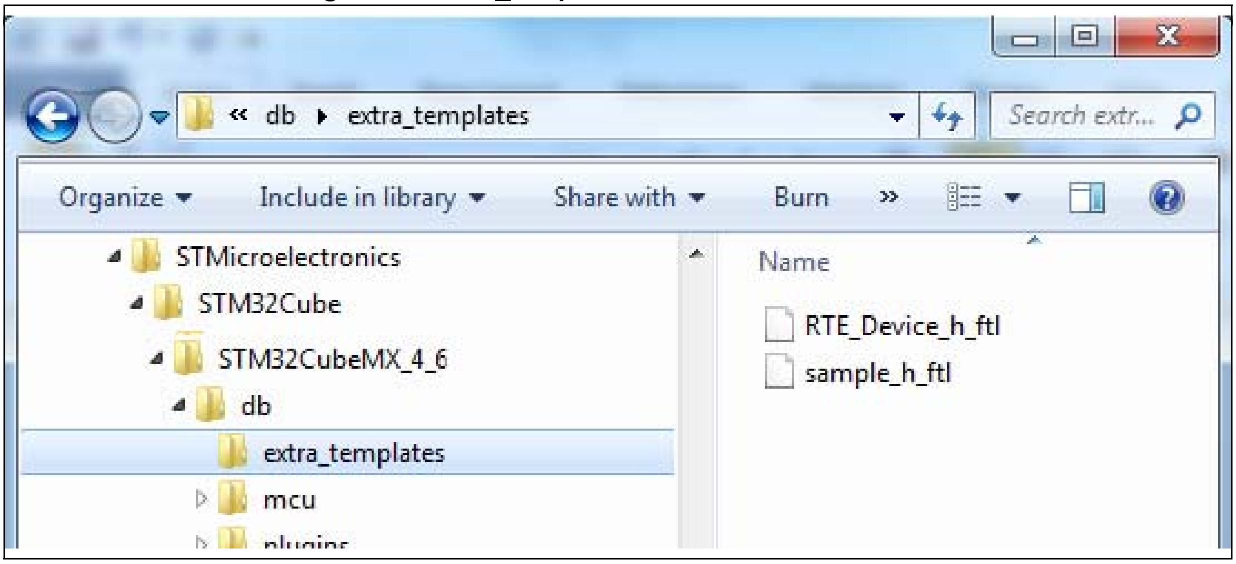



**Figure 71. extra_templates folder with user templates**

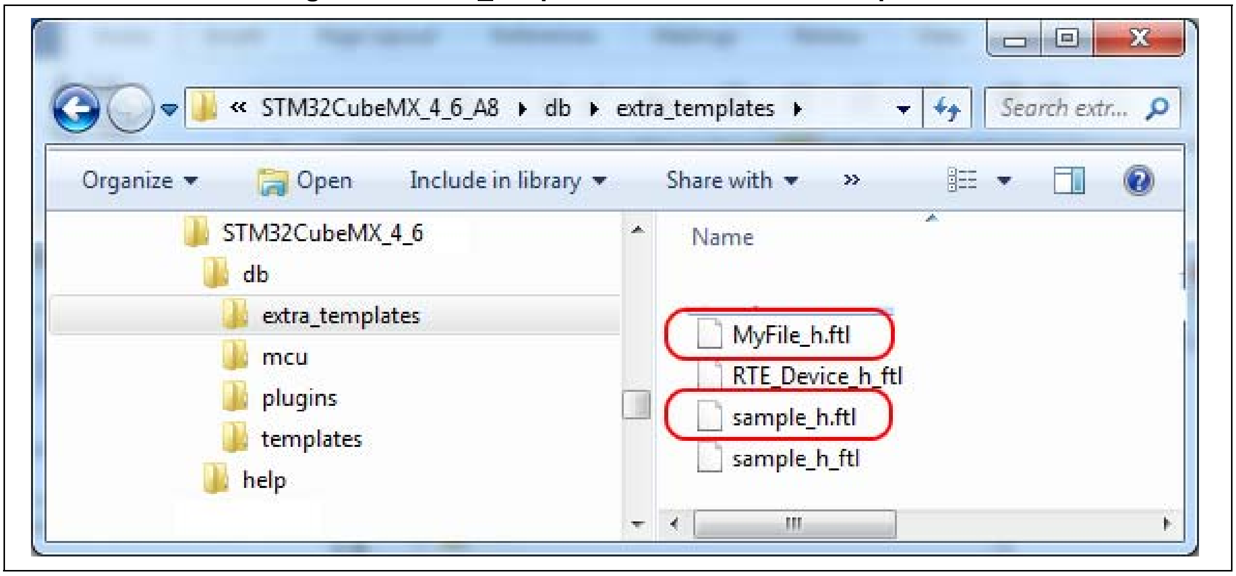


**Figure 72. Project root folder with corresponding custom generated files**

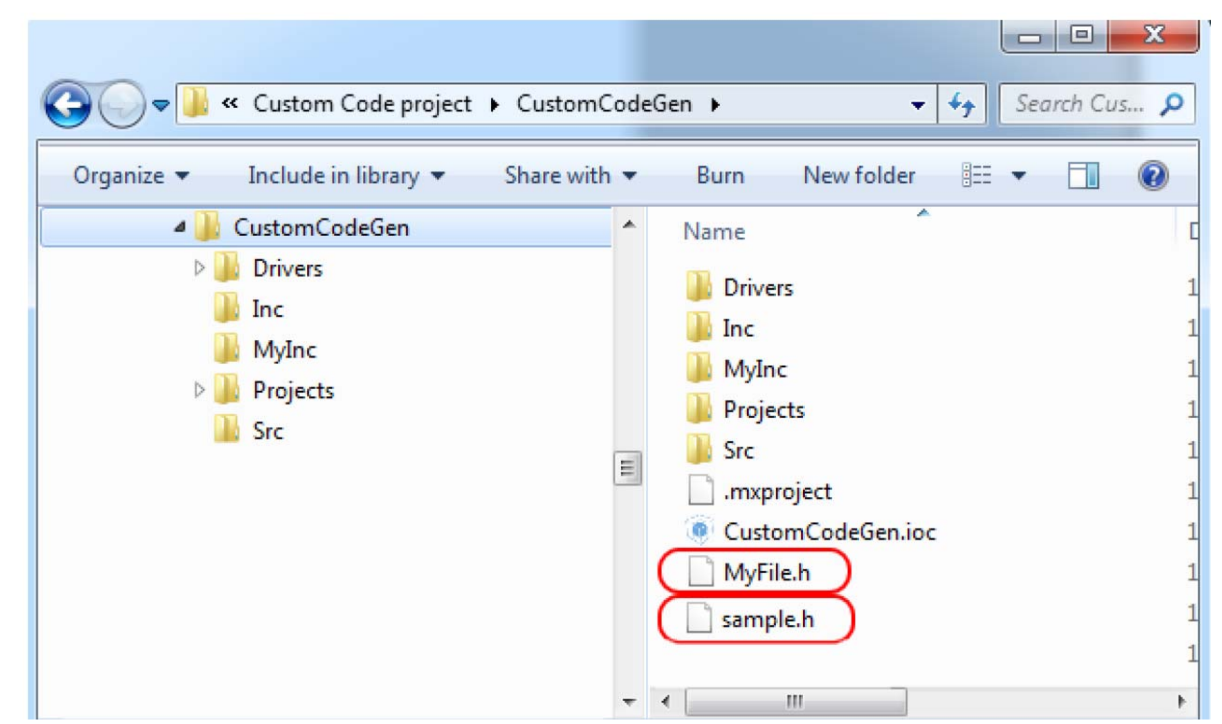



**Figure 73. User custom folder for templates**

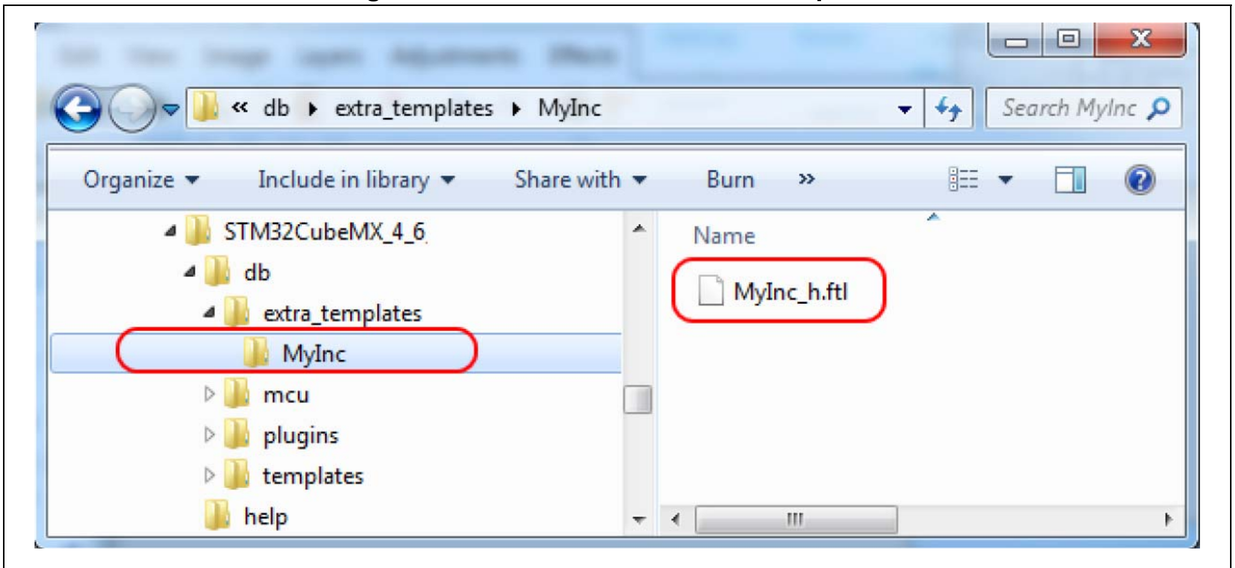
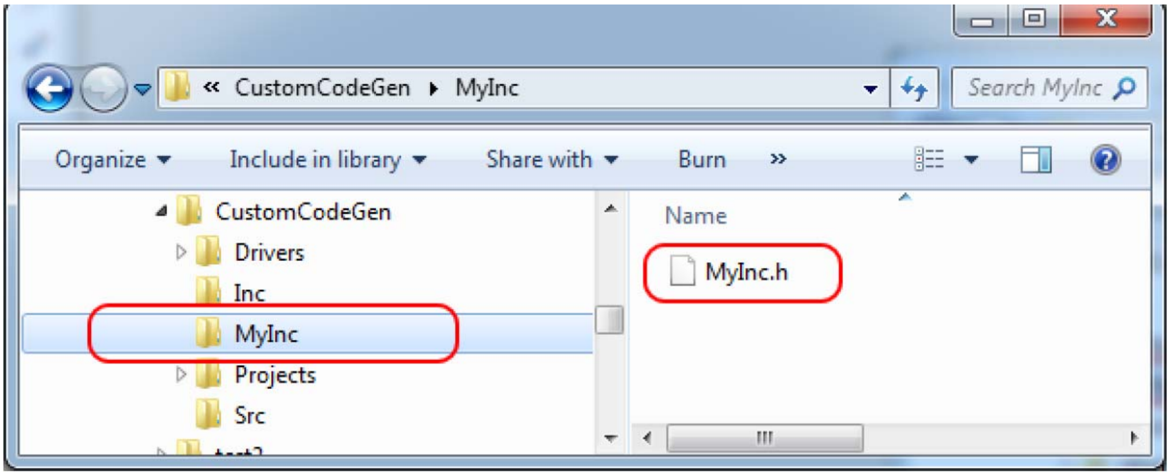


**Figure 74. Custom folder with corresponding custom generated files**



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

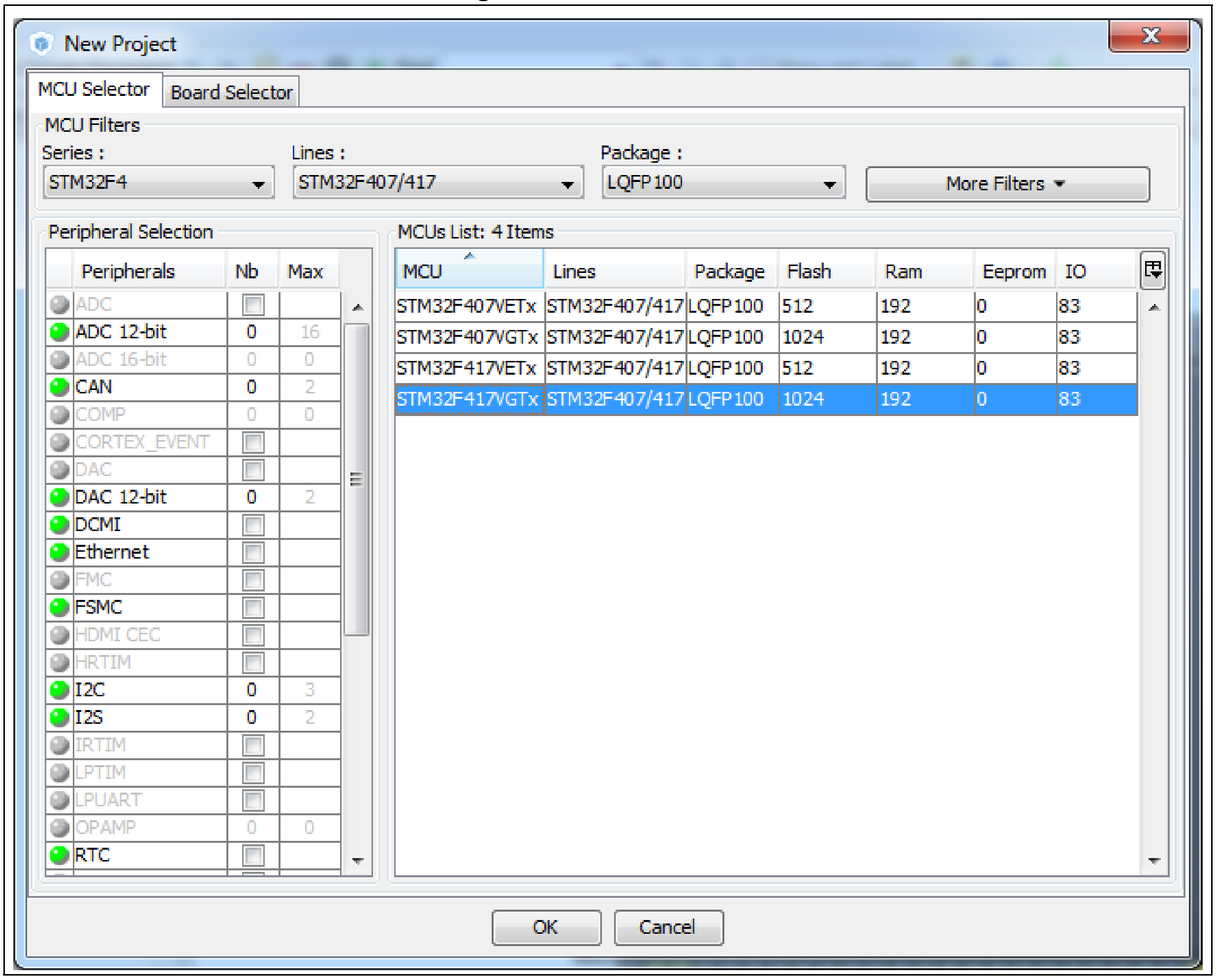
This section describes the configuration and C code generation process. It takes as an example a simple LED toggling application running on the STM32F4DISCOVERY board.

## 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 75*).

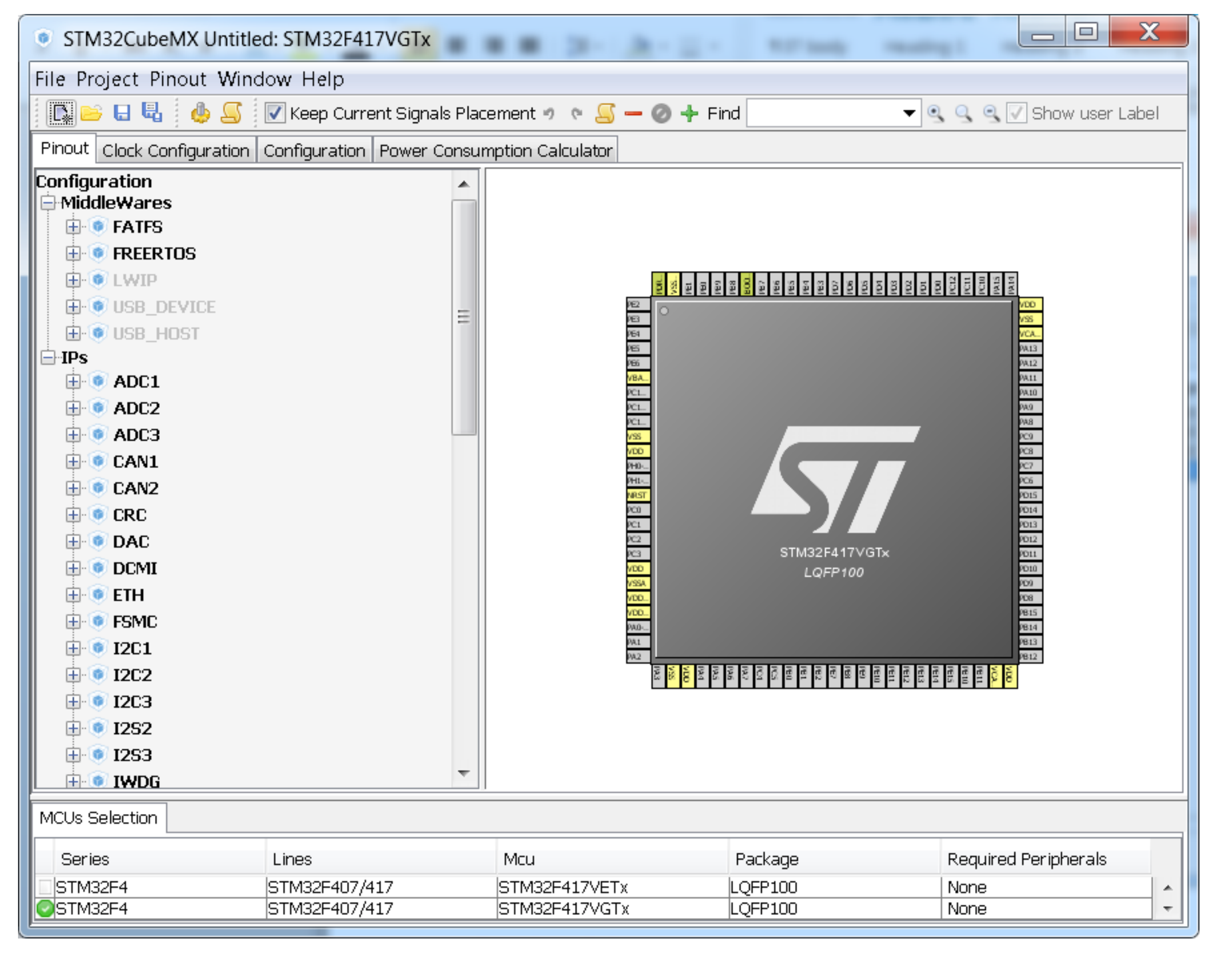
   As an example, you can filter down on STM32F4/STM32F407 and LQFP100 package.
3. Select the MCU Selector tab and filter down.
4. Select the STM32F407VGTx MCU and click OK.

**Figure 75. MCU selection**



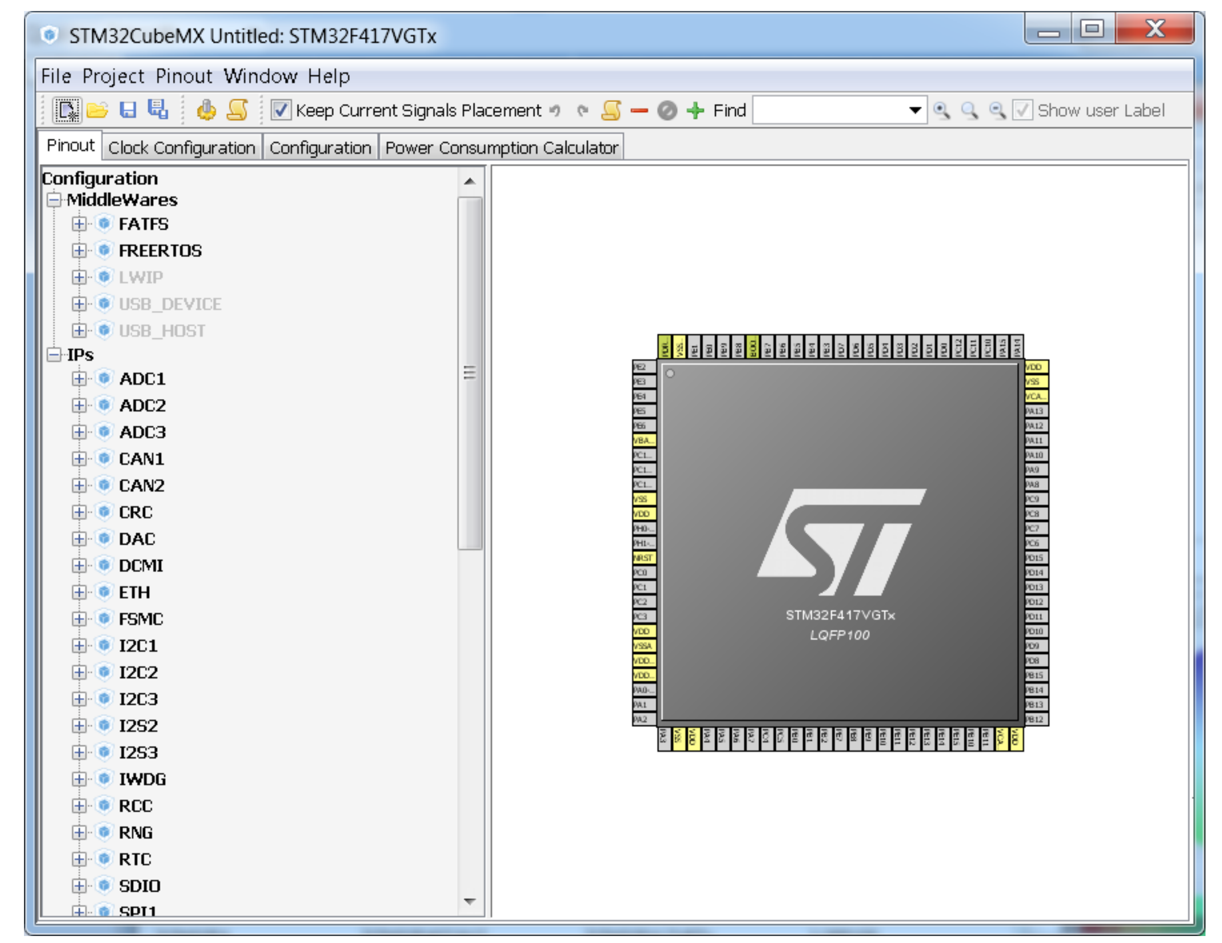
STM32CubeMX views are then populated with the selected MCU database (see *Figure 76*).

**Figure 76. Pinout view with MCUs selection**



Optionally, remove the MCUs Selection bottom window by unselecting **Window> Outputs** sub-menu (see *Figure 77*).

**Figure 77. Pinout view without MCUs selection window**



## 6.2 Configuring the MCU pinout

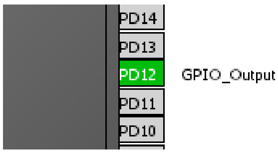
For a detailed description of menus, advanced actions and conflict resolutions, refer to *Section 4: STM32CubeMX User Interface* and *Appendix A: STM32CubeMX pin assignment rules*.

1. By default, STM32CubeMX loads 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.

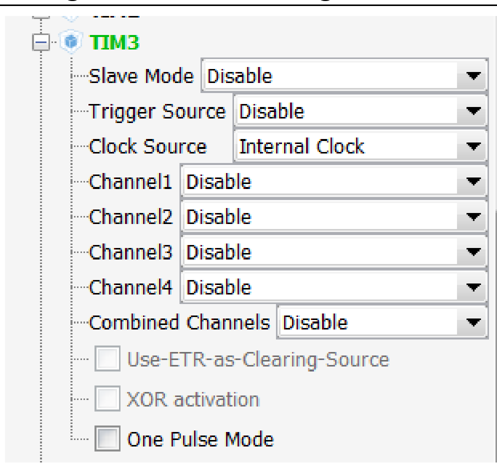
   This setting is saved as a user preference in order to be restored when reopening the tool or when loading another project.
3. Select the required peripherals and peripheral modes:
   a) Configure the GPIO to output the signal on the STM32F4DISCOVERY green LED by right-clicking PD12 from the **Chip** view, then select GPIO_output:

**Figure 78. GPIO pin configuration**




   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 79*).

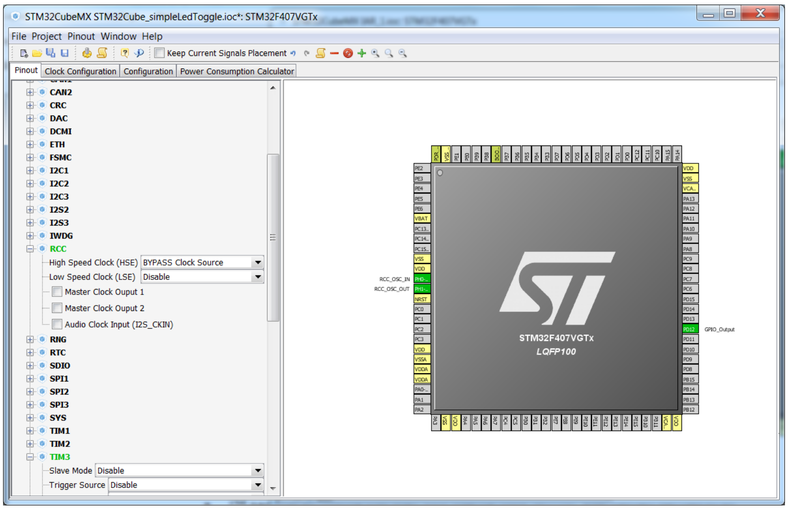
**Figure 79. Timer configuration**



c) You can also configure the RCC in order to use an external oscillator as potential clock source (see *Figure 80*).

This completes the pinout configuration for this example.

**Figure 80. Simple pinout configuration**



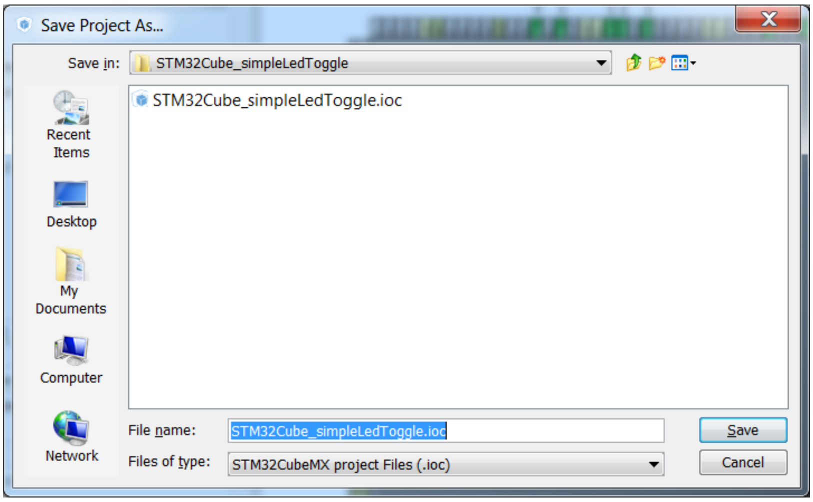
*Note: Starting with STM32CubeMX 4.2, the user can skip the pinout configuration by directly loading ST Discovery board configuration from the Board selector tab.*

## 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 81. Save Project As window**




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 82*). A .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 83*).

**Figure 82. Generate Project Report - New project creation**

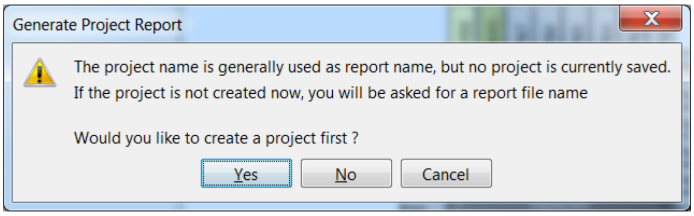
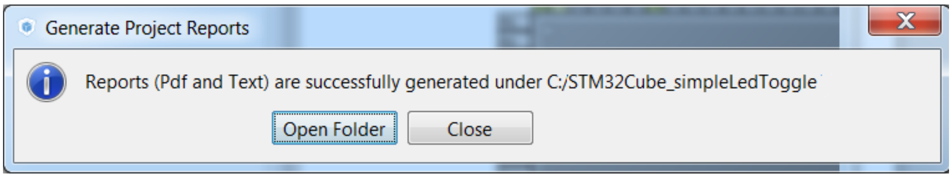



**Figure 83. Generate Project Report - Project successfully created**




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).

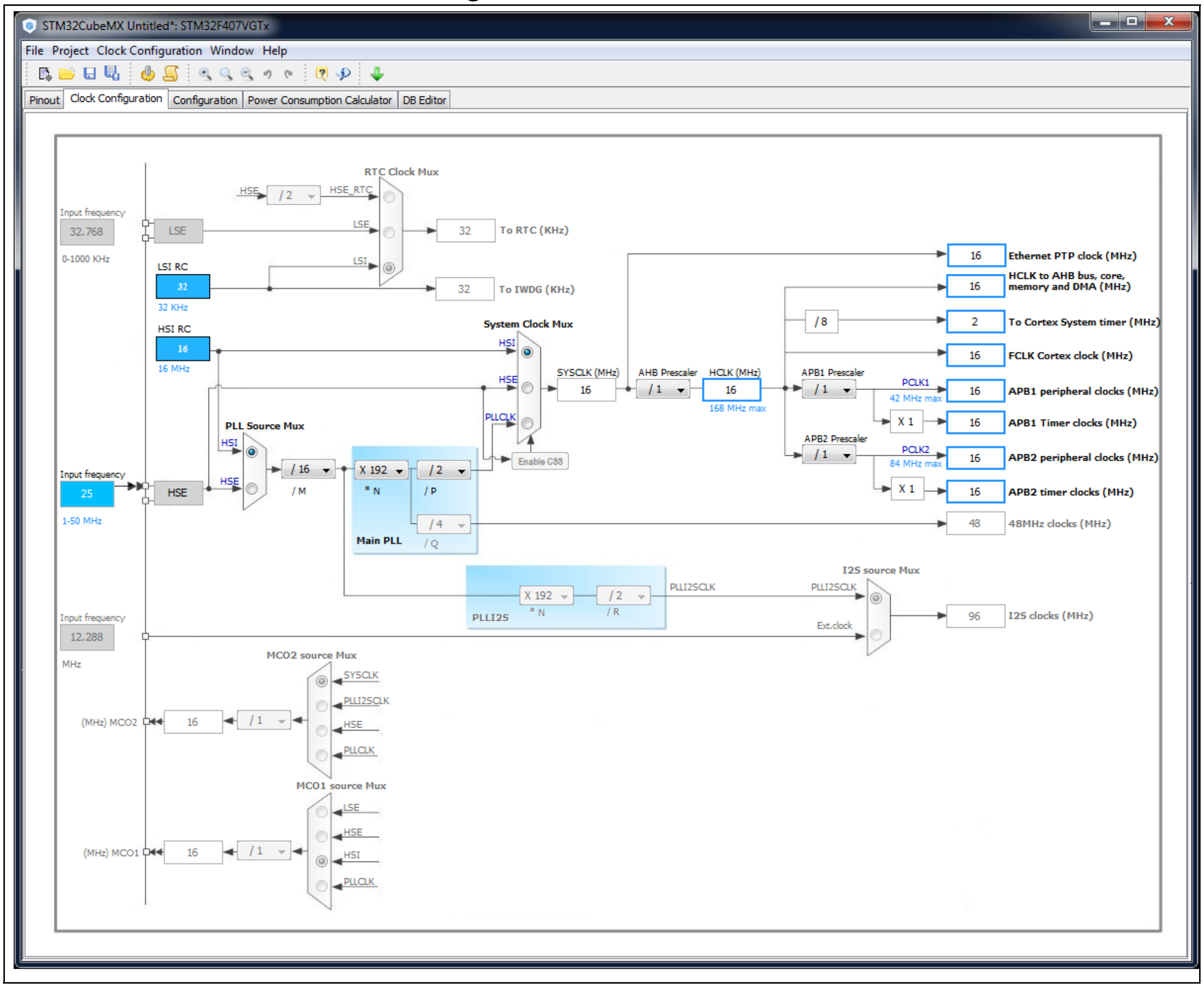
STM32CubeMX uses the clock settings defined in the Clock tree view to generate the initialization C code for each peripheral clock. Clock settings are performed in the generated C code as part of RCC initialization within the project main.c and in stm32f4xx_hal_conf.h (HSE, HSI and External clock values 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 84*).

   The internal (HSI, LSI), system (SYSCLK, HCLK) 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.

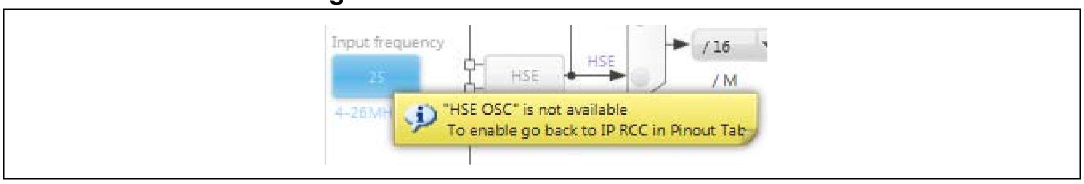
**Figure 84. Clock tree view**




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

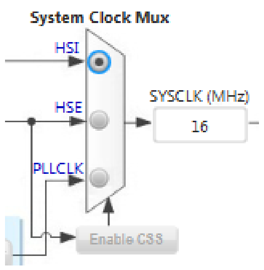
   To use an external clock source (HSE or LSE), the RCC peripheral shall be configured in the **Pinout** view since pins will be used to connect the external clock crystals (see *Figure 85*).

**Figure 85. HSE clock source disabled**



In the example taken for the tutorial, select HSI to use the internal 16 MHz clock (see *Figure 86*).

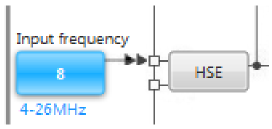
**Figure 86. HSI clock enabled**




Other options would have been:

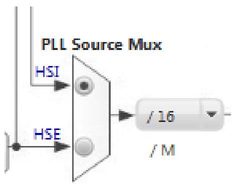
– To select the external HSE source and enter 8 in the HSE input frequency box since an 8 MHz crystal is connected on the discovery board:

**Figure 87. HSE clock source enabled**




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

**Figure 88. 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. Click to generate the corresponding clock initialization C code or 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 STM32CubeF4 firmware libraries.

For all other STM32 series, STM32CubeMX generates only the initialization C code corresponding to the MCU pin configuration using the STM32 standard peripheral libraries.

### 6.6.1 Initial conditions

Select the **Configuration** tab to display the configuration view (see *Figure 89*).

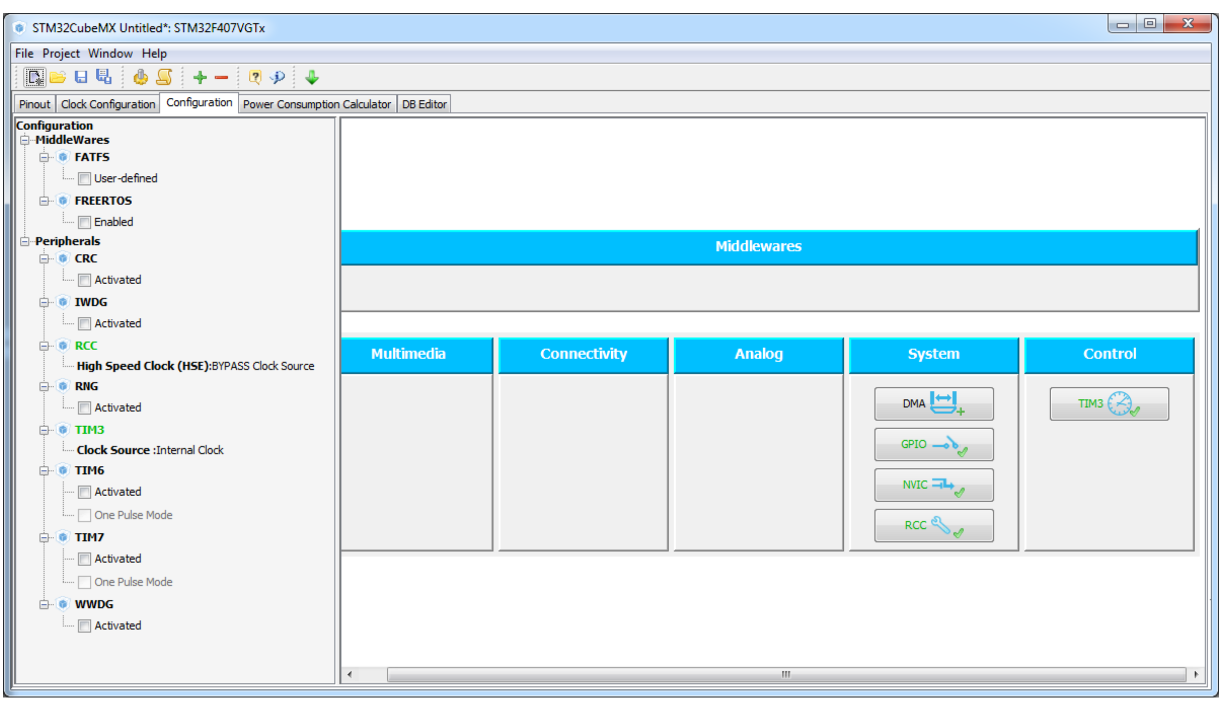
In this view, it is not possible to configure the peripherals to operate in a mode influencing the pinout. This can be done by using only the **Pinout** view.

Peripherals and middleware modes without influence on the pinout can be disabled or enabled in the IP Tree panel.

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 89. Configuration view**

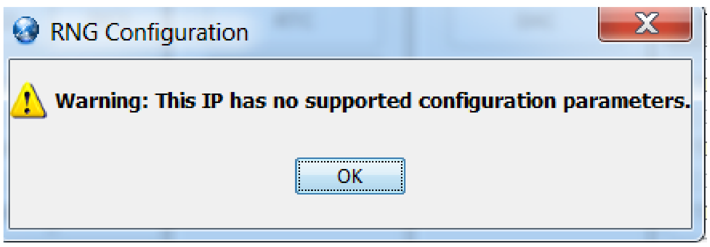



## 6.6.2 Configuring the peripherals

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

Some peripheral modes have no configurable parameters as illustrated below:

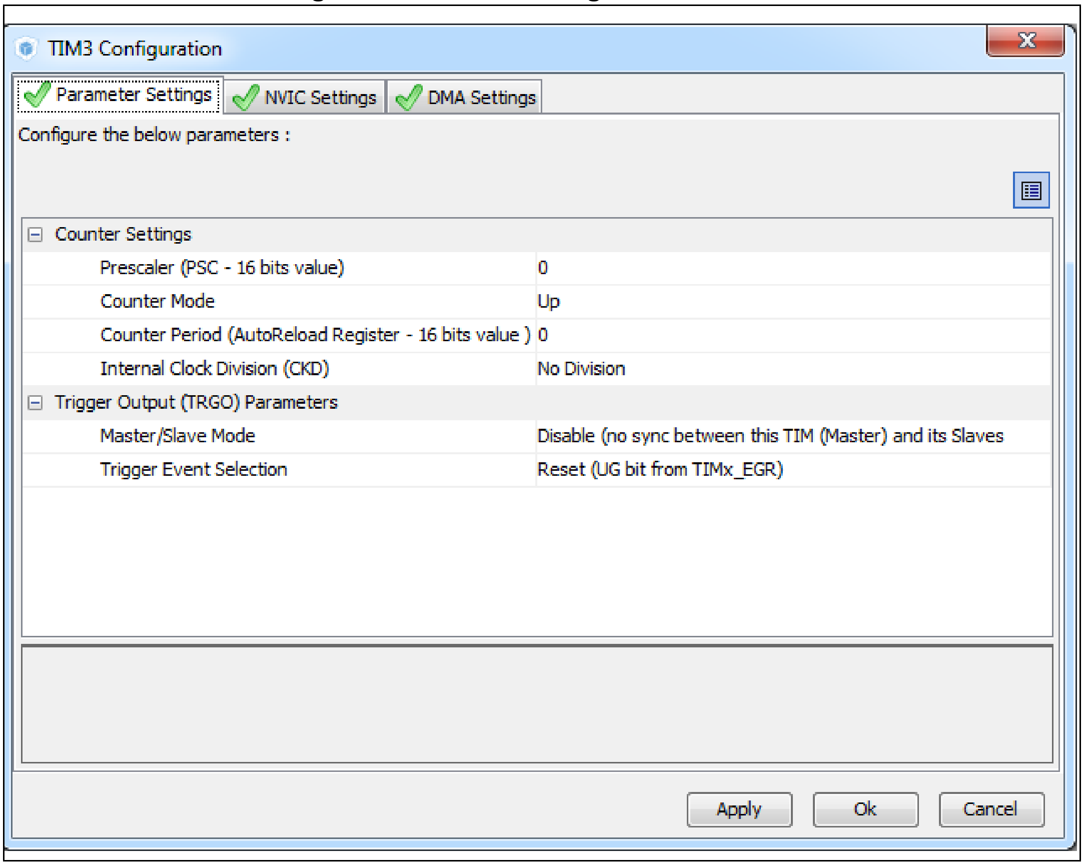
**Figure 90. Case of IP without configuration parameters**




Follow the steps below to proceed with peripheral configuration:

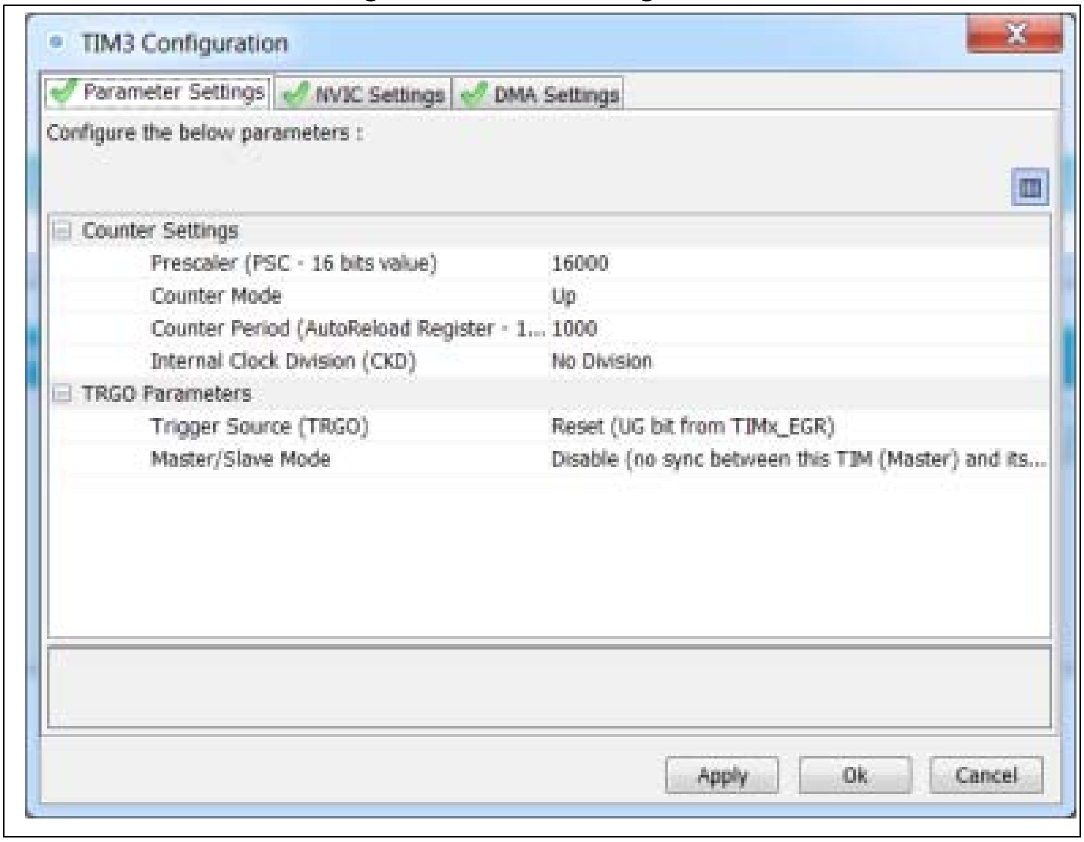
1. Click the peripheral button to open the corresponding configuration window.
   In our example,
   a) Click TIM3 to open the timer configuration window.

**Figure 91. Timer 3 configuration window**



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 92. Timer 3 configuration**




2. Optionally and when available, select

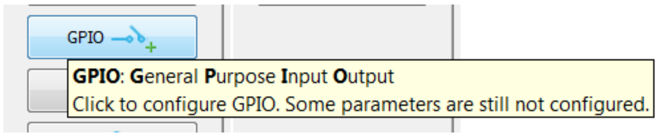
- The **NVIC Settings** tab to display the NVIC configuration and enable interruptions for this peripheral.
- The **DMA Settings** tab to display the DMA configuration and to configure DMA transfers for this peripheral.
- The **GPIO Settings** tab to display the GPIO configuration and to configure the GPIOs for this peripheral.

3. Modify and click Apply or OK to save your modifications.
4. Click Cancel and OK to exit the **Configuration** window.

### 6.6.3 Configuring the GPIOs

The user can adjust all pin configurations from this window. Color scheme (black label, + sign) and tooltip indicate that the GPIO configuration is incomplete:

**Figure 93. GPIO configuration color scheme and tooltip**

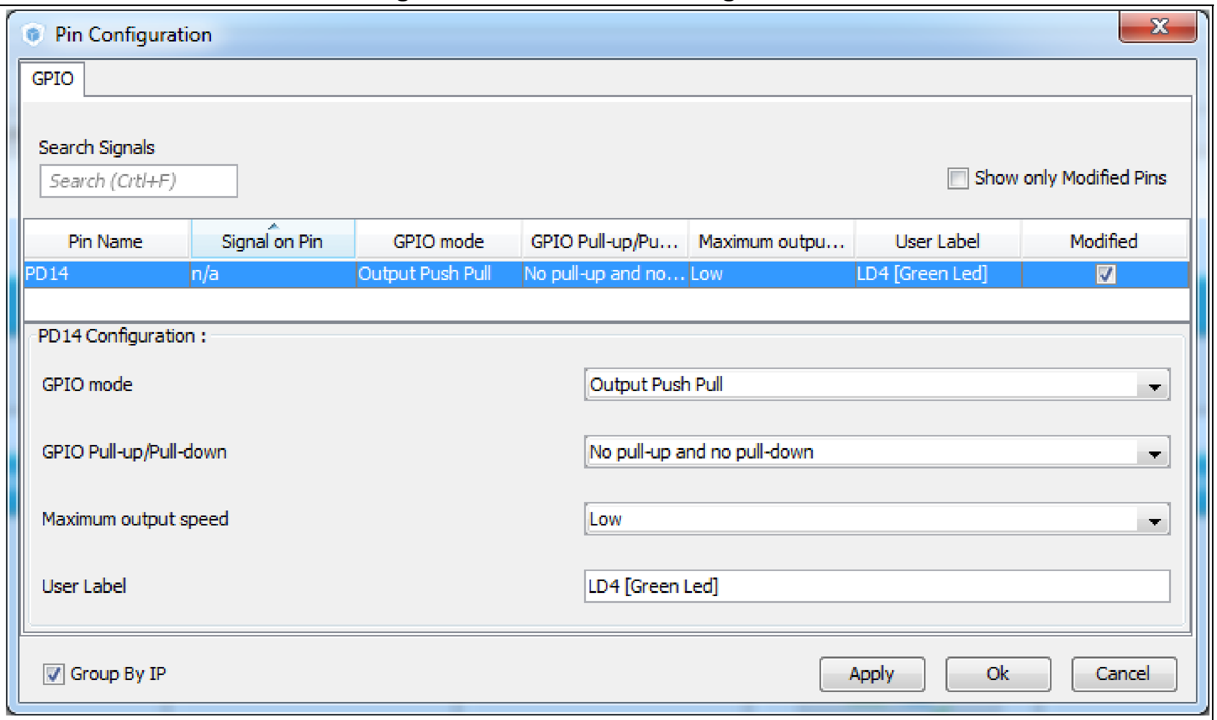



Follow the sequence below to configure the GPIOS:

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 94*).

**Figure 94. GPIO mode configuration**




3. Click Ok to close the window.

### 6.6.4 Configuring the DMAs

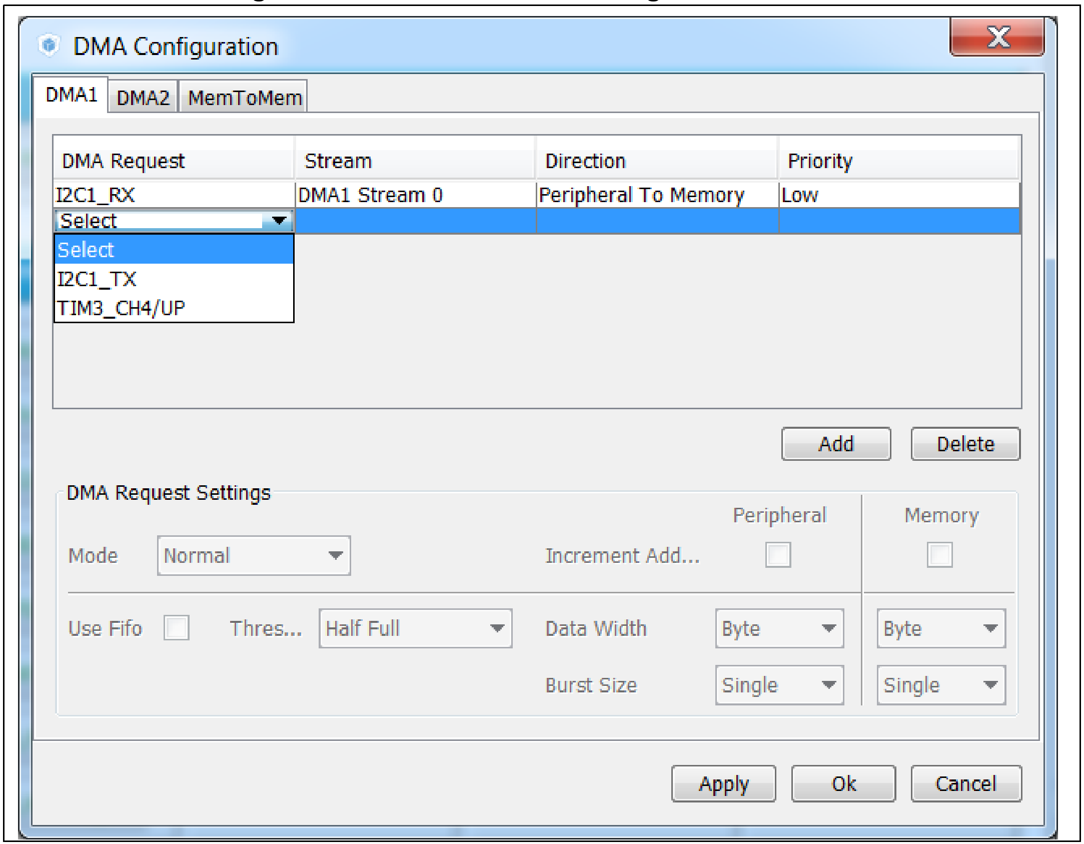
This is not required for the example taken for the tutorial.

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 95*).

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 95. DMA Parameters configuration window**




### 6.6.5 Configuring the middleware

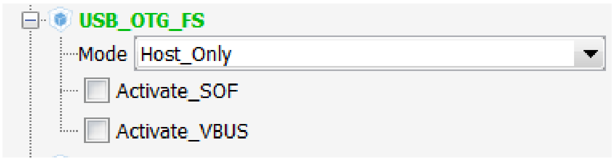
This is not required for the example taken for the tutorial.

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 96. FATFS disabled**


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

**Figure 97. USB Host configuration**

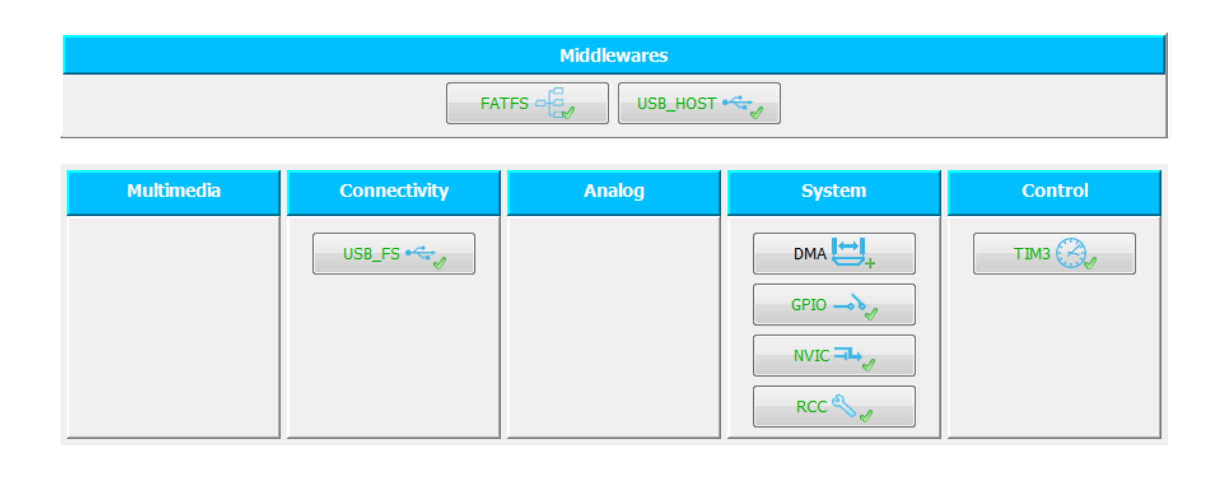



2. Select MSC_FS class from USB Host middleware.
3. Select the checkbox to enable FATFS USB mode in the tree panel.

**Figure 98. FATFS over USB mode enabled**


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

**Figure 99. Configuration view with FATFS and USB enabled**




5. FATFS and USB using default settings are already marked as configured. Click FATFS and USB buttons to display default configuration settings. You can also change them by following the guidelines provided at the bottom of the window.

**Figure 100. FATFS IP instances**

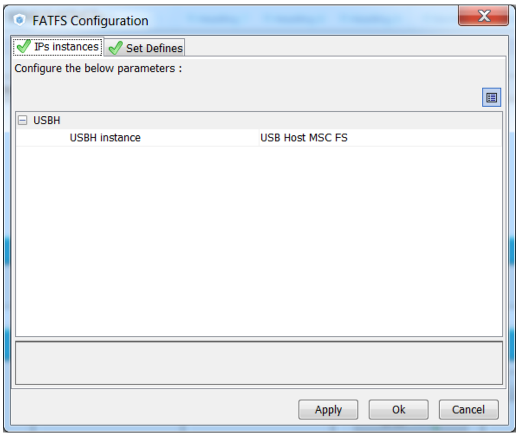
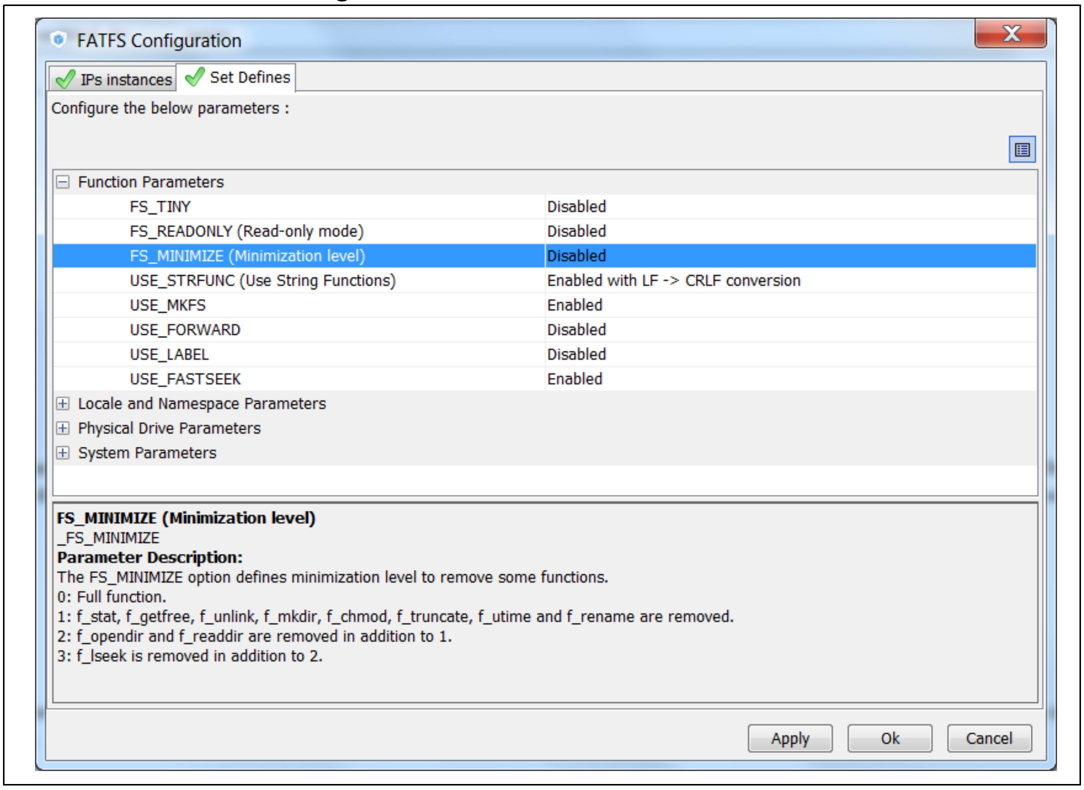


**Figure 101. FATFS define statements**



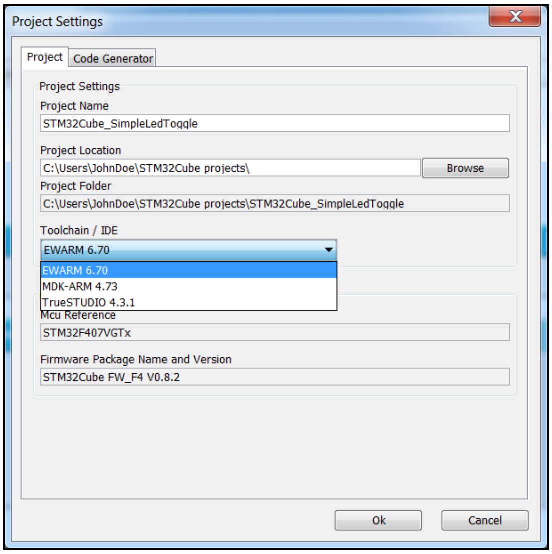
## 6.7 Generating a complete C project

### 6.7.1 Setting project options

Default project settings can be adjusted prior to C code generation as described in *Figure 102*.

1. Select **Settings** from the **Project** menu to open the Project settings window.
2. Select the **Project Tab** and choose a Project **name**, **location** and a **toolchain** to generate the project (see *Figure 102*).

**Figure 102. Project Settings and toolchain choice**

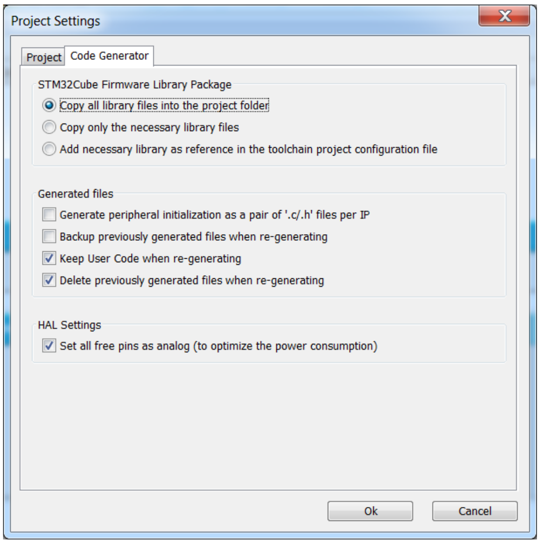



3. Select the **Code Generator** tab to choose various C code generation options:
   - The library files copied to *Projects* folder.
   - C code regeneration (e.g. what is kept or backed up during C code regeneration).
   - HAL specific action (e.g. set all free pins as analog I/Os to reduce MCU power consumption).

   In the tutorial example, select the settings as displayed in the figure below and click OK.

*Note: A dialog window appears when the firmware package is missing. Go to next section for explanation on how to download the firmware package.*

**Figure 103. Project Settings menu - Code Generator tab**

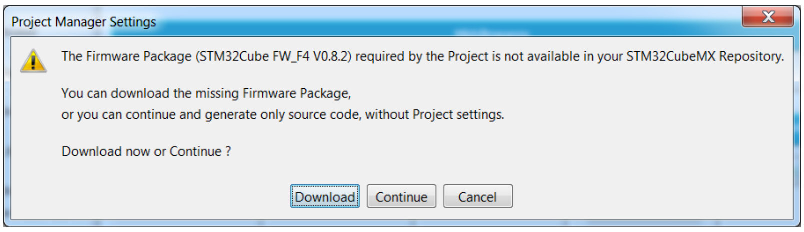



## 6.7.2 Downloading firmware package and generating the C code

1. Click to generate the C code.

   During C code generation, STM32CubeMX copies files from the relevant STM32Cube firmware package into the project folder so that the project can be compiled. When generating a project for the first time, the firmware package is not available on the user PC and a warning message is displayed:

**Figure 104. Missing firmware package warning message**

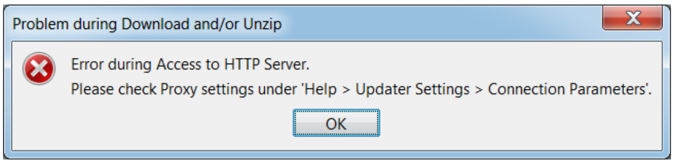



2. STM32CubeMX offers to download the relevant firmware package or to go on. Click **Download** to obtain a complete project, that is a project ready to be used in the selected IDE.

   By clicking **Continue**, only Inc and *Src* folders will be created, holding STM32CubeMX generated initialization files. The necessary firmware and middleware libraries will have to be copied manually to obtain a complete project.

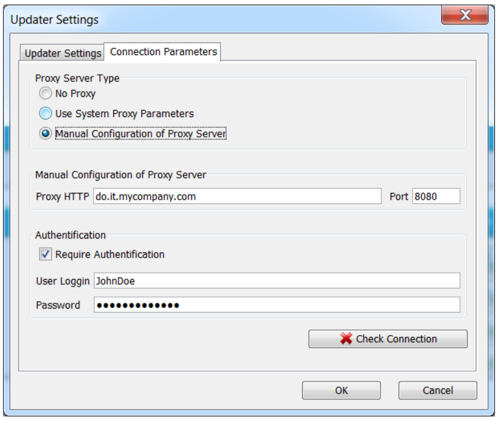
   If the download fails, the below error message is displayed :

**Figure 105. Error during download**



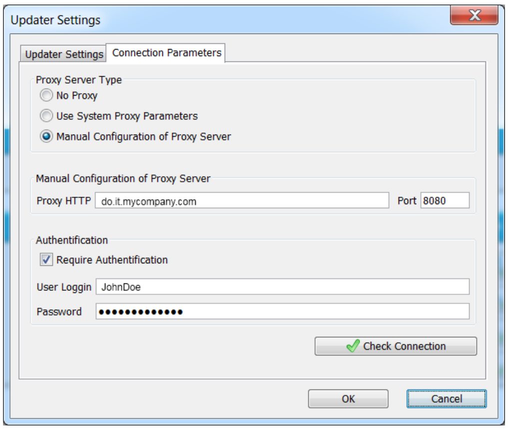

3. Select **Help > Updater settings menu** and adjust the connection parameters to match your network configuration.

**Figure 106. Updater settings for download**




4. Click **Check connection.** The check mark turns green once the connection is established.

**Figure 107. Updater settings with connection**



5. Once the connection is functional, click to generate the C code. The C code generation process starts and progress is displayed as illustrated in the next figures.

**Figure 108. Downloading the firmware package**

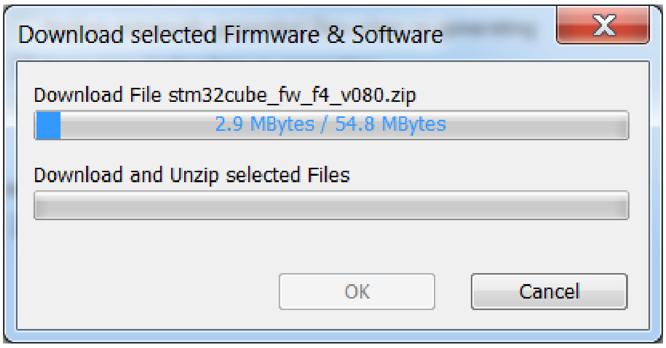
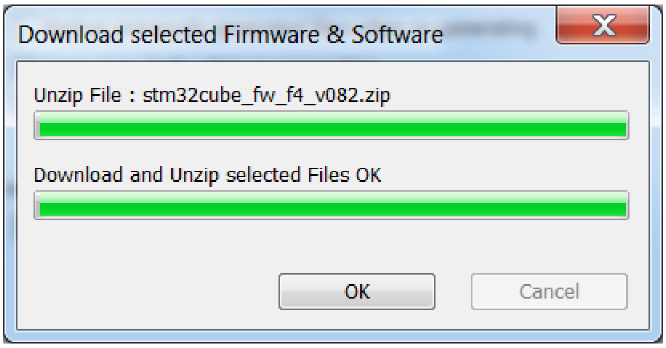
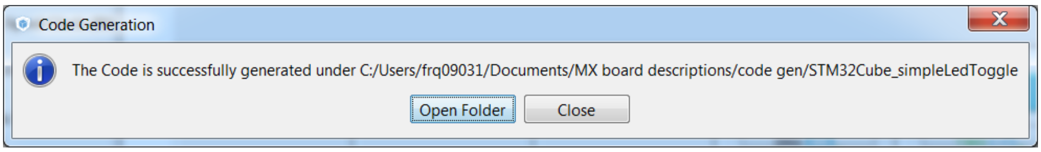



**Figure 109. Unzipping the firmware package**



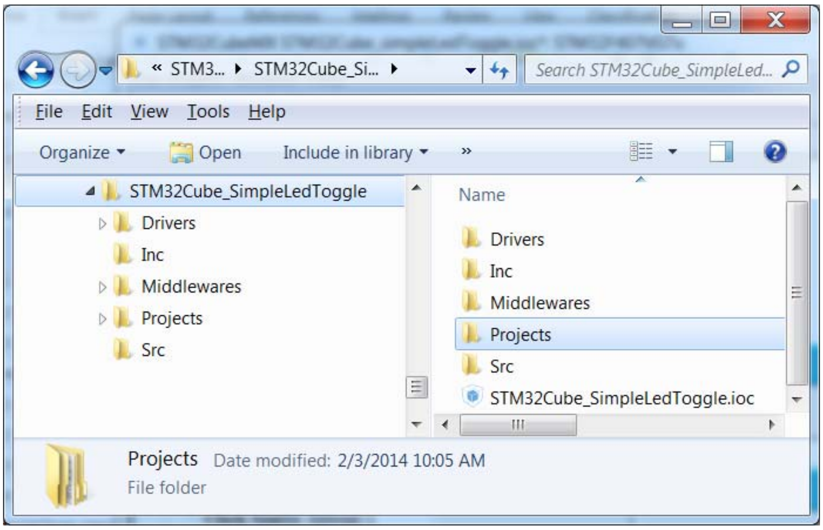

6. Finally, a confirmation message is displayed to indicate that the C code generation has been successful.

**Figure 110. C code generation completion message**



7. Click **Open Folder** to display the generated project contents.

**Figure 111. C code generation output folder**




When generated project contains:

- The STM32CubeMX .ioc project file located in the root folder. It contains the project user configuration and settings generated through STM32CubeMX user interface.
- The *Drivers* and *Middlewares* folders hold copies of the firmware package files relevant for the user configuration.
- The *Projects* folder contains IDE specific folders with all the files required for the project development and debug within the IDE.
- The *Inc* and *Src* folders contain STM32CubeMX generated files for middleware, peripheral and GPIO initialization, including the main.c file. The STM32CubeMX generated files contain user-dedicated sections allowing to insert user-defined C code.

**Caution:** C code written within the user sections is preserved at next C code generation, while C code written outside these sections is overwritten.

User C code will be lost if user sections are moved or if user sections delimiters are renamed.

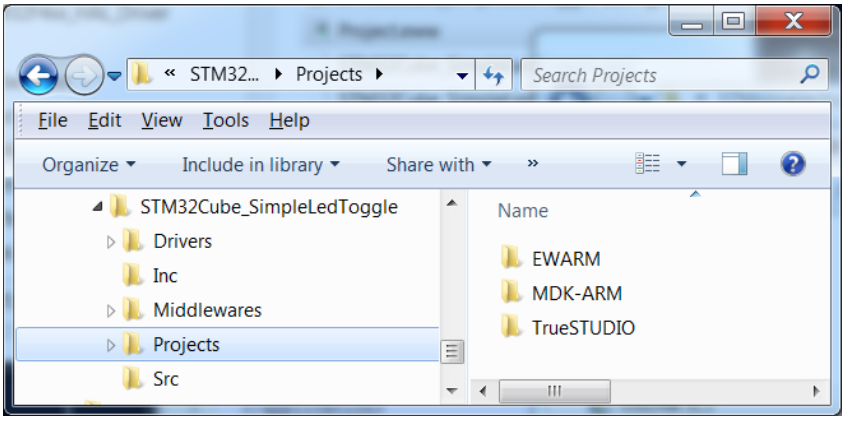
## 6.8 Building and updating the C code project

This example explains how to use the generated initialization C code and complete the project, within IAR EWARM toolchain, to have the LED blink according to the TIM3 frequency.

1. Open the **Projects** folder.

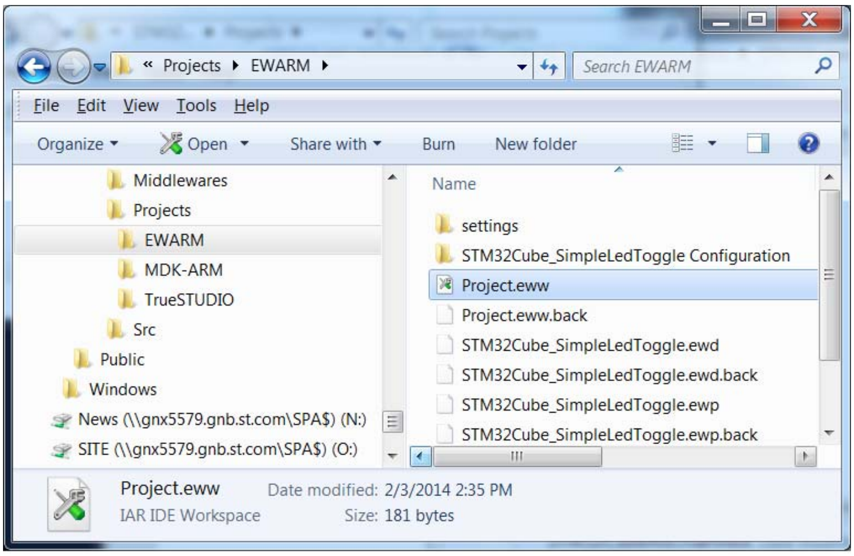
   A folder is available for the toolchains selected for C code generation: the project can be generated for more than one toolchain by choosing a different toolchain from the Project Settings menu and clicking Generate code once again.

**Figure 112. C code generation output: Projects folder**



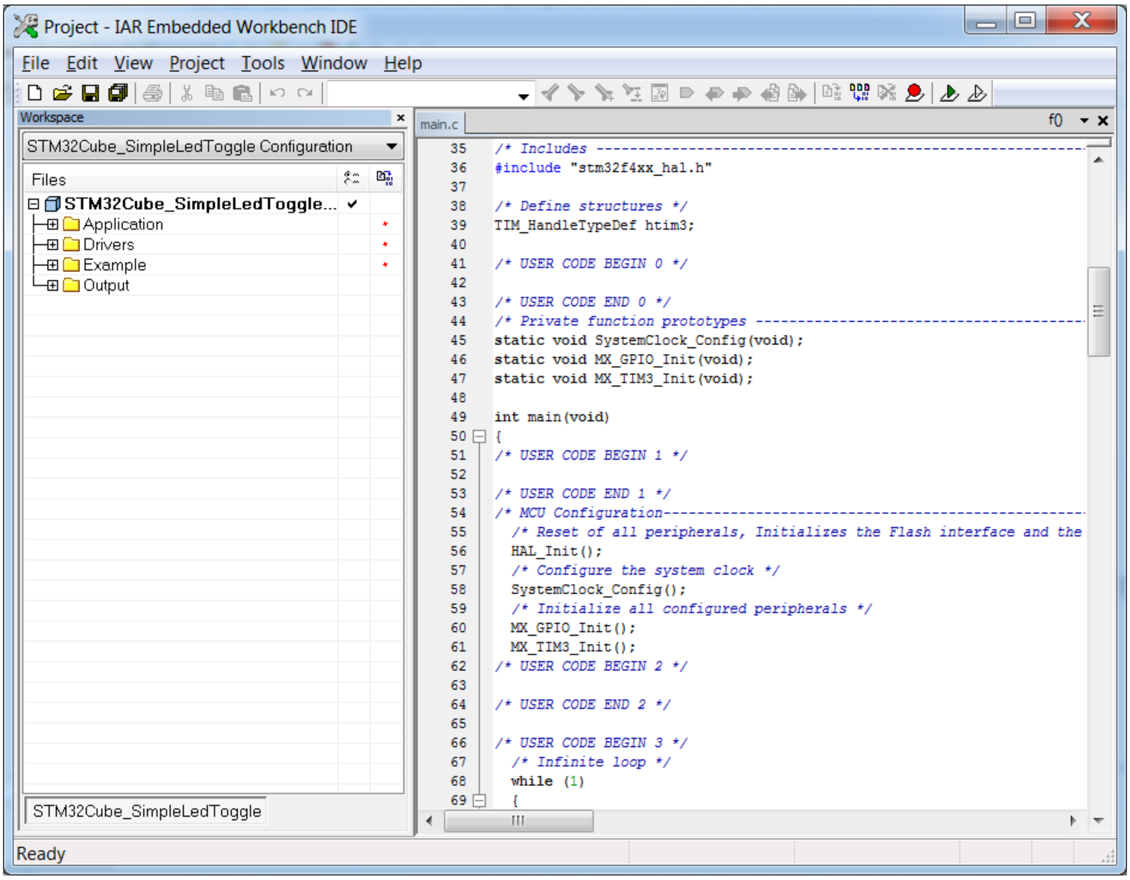

2. Browse to the toolchain to be selected and select the IDE workspace file to open the project directly in the chosen IDE. As an example, select .eww file to load the project in the IAR EWARM IDE.

**Figure 113. C code generation for EWARM**



3. Select the main.c file to open in editor.

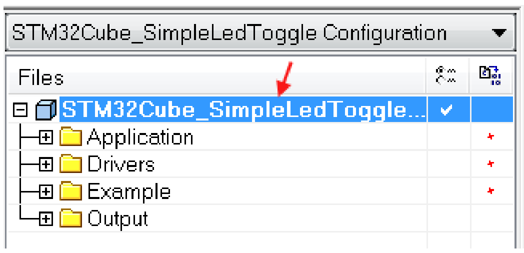
**Figure 114. STM32CubeMX generated project open in IAR IDE**




The htim3 structure handler, system clock, GPIO and TIM3 initialization functions are defined. The initialization functions are called in the main.c. For now the user C code sections are empty.

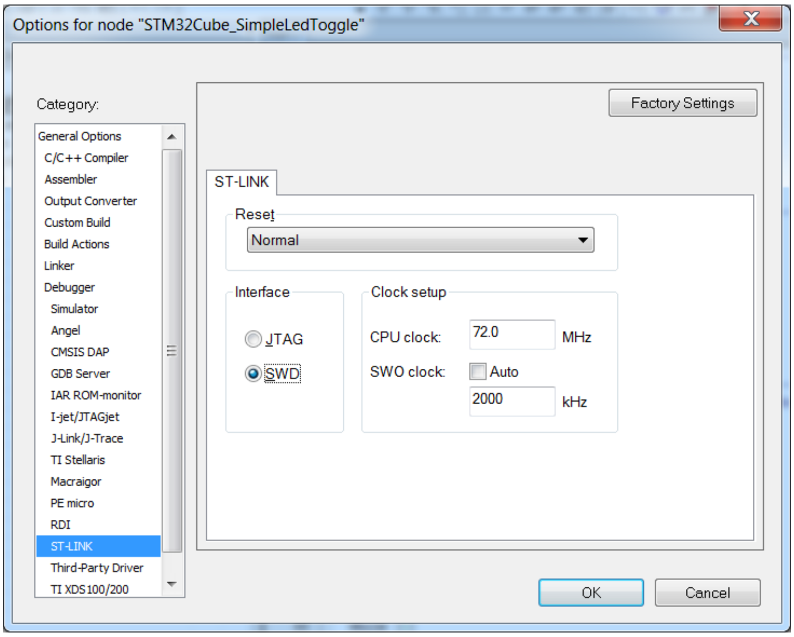
4. In the IAR IDE, right-click the project name and select **Options**.

**Figure 115. IAR options**



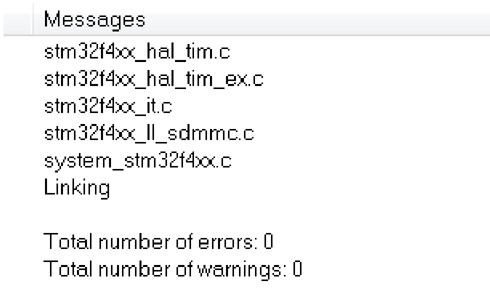
5. Click the ST-LINK category and select SWD for communication with STM32F4DISCOVERY board. Click OK.

**Figure 116. SWD connection**




6. Select **Project > Rebuild all**. Check if the project building has succeeded.

**Figure 117. Project building log**




7. Add user C code in the dedicated user sections **only**.

*Note: The main while(1) loop is placed in a user section.*

For example:

a) Edit the main.c file.

b) To start timer 3, update User Section 3 with the following C code:

**Figure 118. User Section 3**

```
/* USER CODE BEGIN 3 */

  HAL_TIM_Base_Start_IT(&htim3);

  /* Infinite loop */
  while (1)
  {

  }
/* USER CODE END 3 */
```

c) Then, add the following C code in User Section 4:

**Figure 119. User Section 4**

```
/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if ( htim->Instance == htim3.Instance )
  {
    HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12);
  }
}
/* USER CODE END 4 */
```

This C code implements the weak callback function defined in the HAL timer driver (stm32f4xx_hal_tim.h) to toggle the GPIO pin driving the green LED when the timer counter period has elapsed.

8. Rebuild and program your board using . Make sure the SWD ST-LINK option is checked as a Project options otherwise board programming will fail.
9. Launch the program using . The green LED on the STM32F4DISCOVERY board will blink every second.
10. To change the MCU configuration, go back to STM32CubeMX user interface, implement the changes and regenerate the C code. The project will be updated, preserving the C code in the user sections if Keep Current Signals Placement option in Project Settings is enabled.

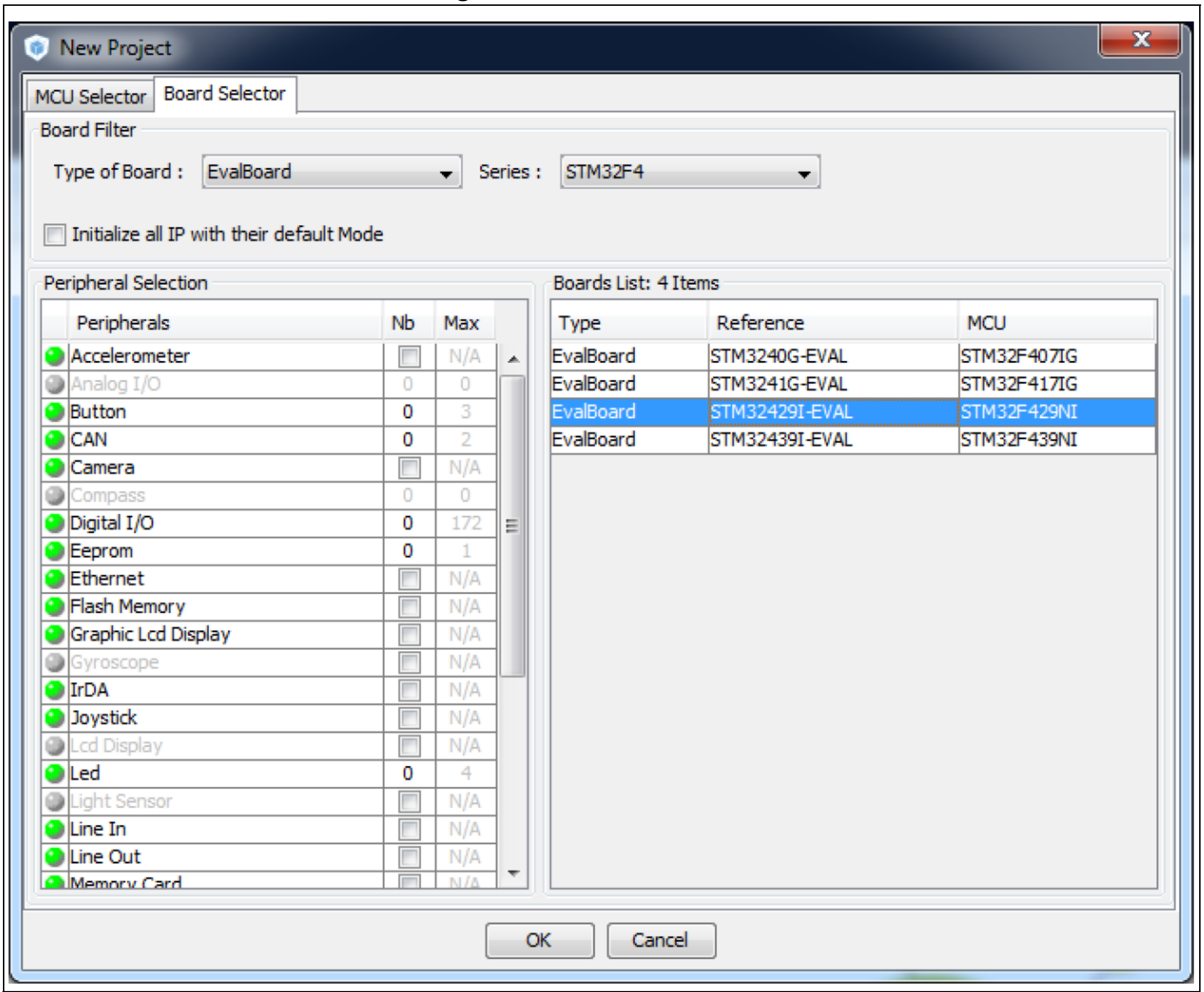
# 7 Tutorial 2 - Example of FATFS on an SD card using STM32F429I-EVAL evaluation board

The tutorial consists in creating and writing to a file on the STM32F429I-EVAL SD card using the FATFS file system middleware.

To generate a project and run tutorial 2, follow the sequence below:

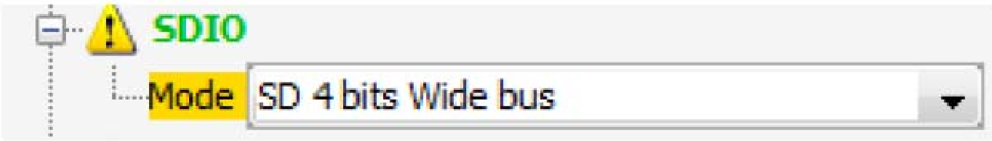
1. Launch STM32CubeMX.
2. Select **File > New Project**. The Project window opens.
3. Click the **Board Selector** Tab to display the list of ST boards.
4. Select **EvalBoard** as type of Board and **STM32F4** as series to filter down the list.
5. Leave the option **Initialize all IPs with their default mode** unchecked so that the code is generated only for the IPs used by the application.
6. Select the STM32F429I-EVAL board and click OK. The **Pinout** view is loaded, matching the MCU pinout configuration on the evaluation board (see *Figure 120*).

**Figure 120. Board selection**



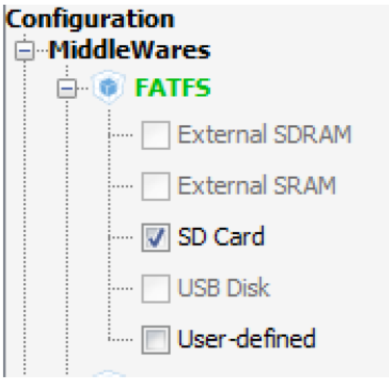
7. From the Peripheral tree on the left, expand the SDIO IP and select the SD 4 bits wide bus (see *Figure 121*).

**Figure 121. SDIO IP configuration**



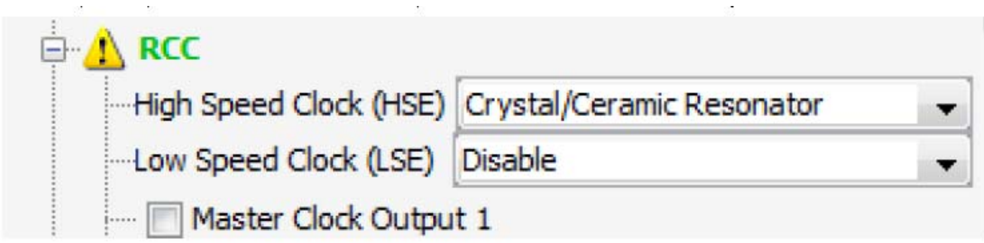

8. Under the Middlewares category, check “SD Card” as FATFS mode (see *Figure 122*).

**Figure 122. FATFS mode configuration**



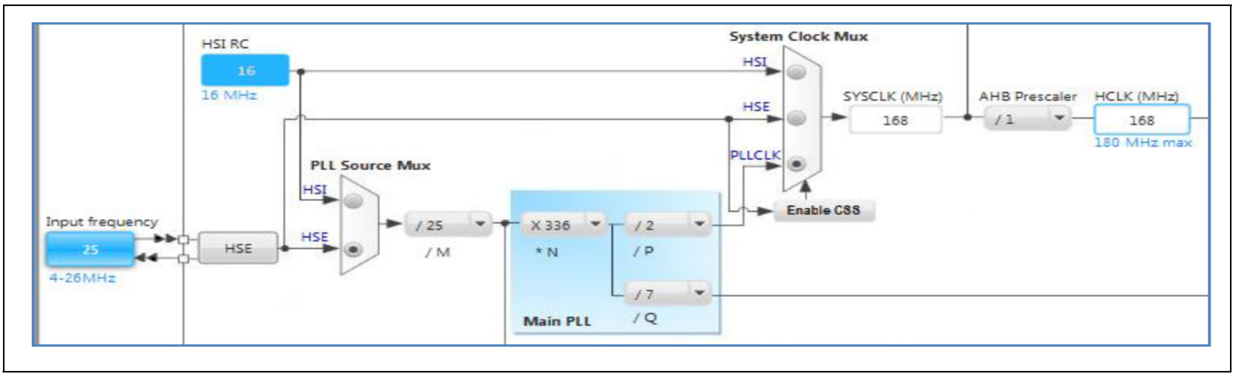

9. Configure the clocks as follows:
   a) Select the RCC peripheral from the **Pinout** view (see *Figure 123*).

**Figure 123. RCC peripheral configuration**



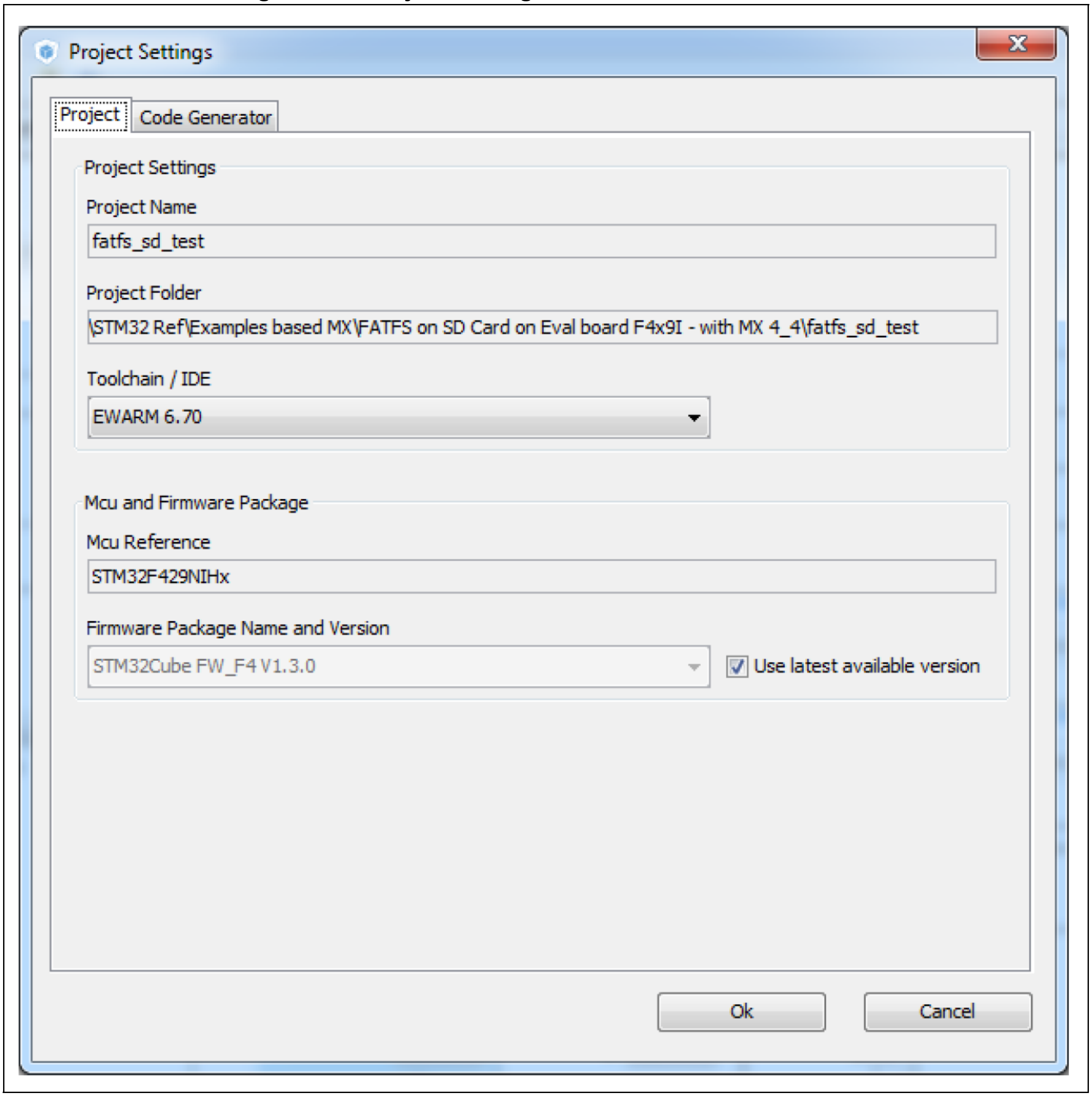
b) Configure the clock tree from the clock tab (see *Figure 124*).

**Figure 124. Clock tree view**




10. In the **Project Settings** menu, specify the project name and destination folder. Then, select the EWARM IDE toolchain.

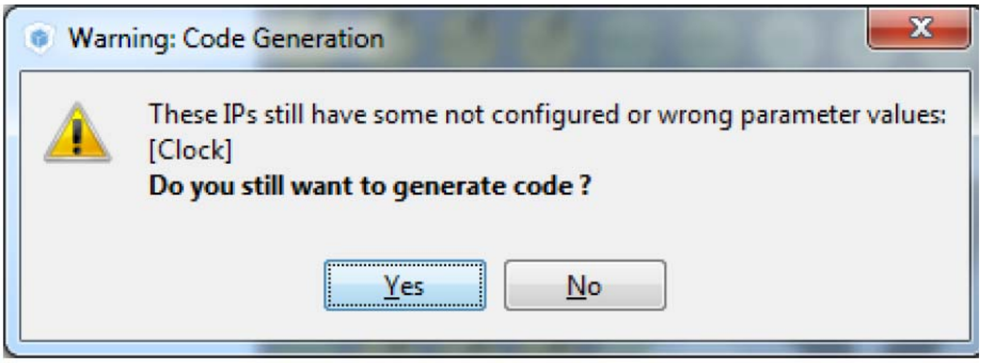
**Figure 125. Project Settings menu - Code Generator tab**




11. Click Ok. Then, in the toolbar menu, click to generate the project.

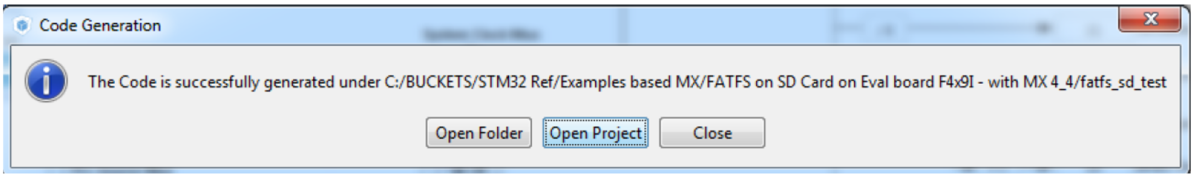
12. Ignore the warning message and click Yes. Code generation progress bar is displayed.

**Figure 126. Code generation error message**



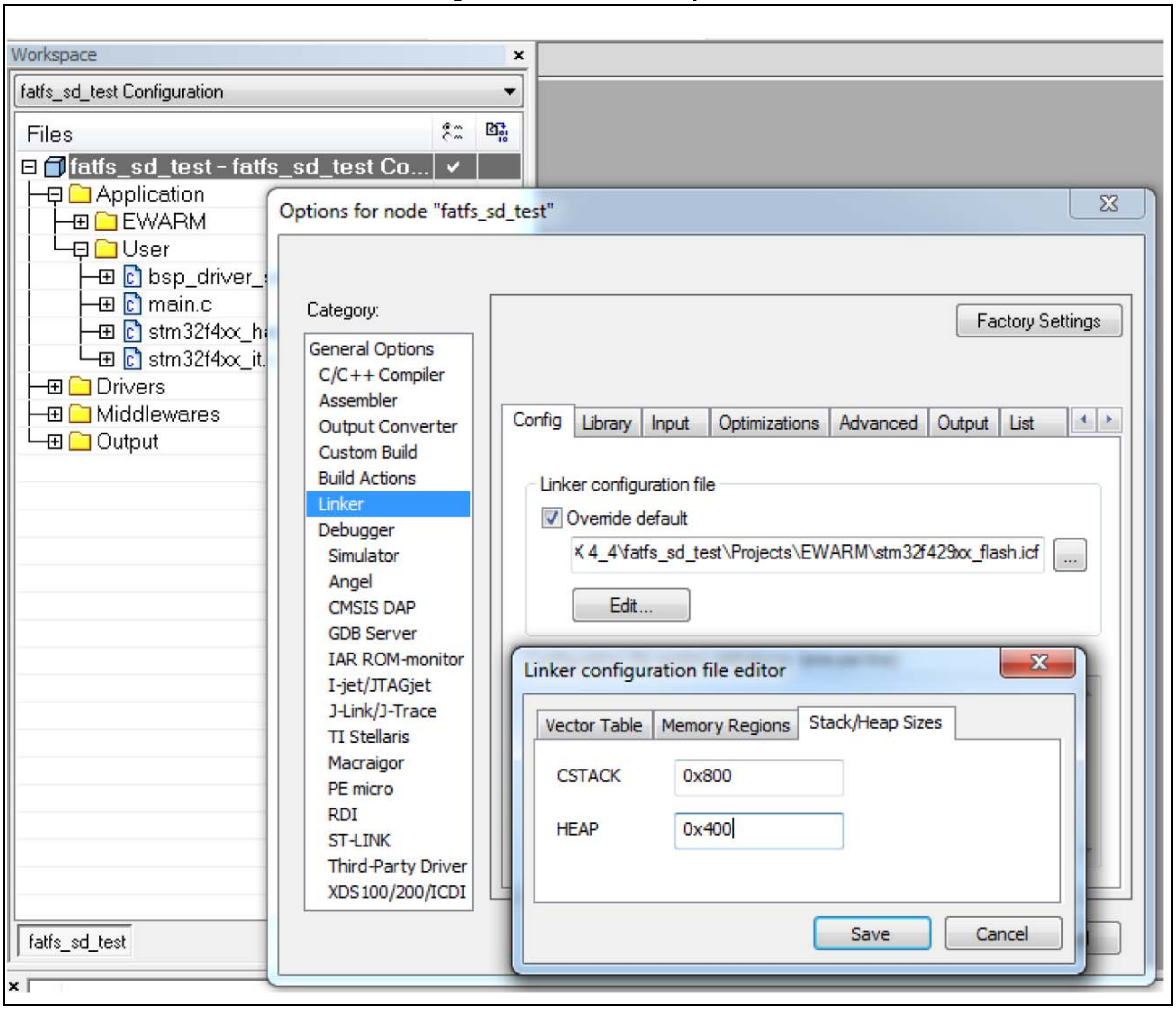

13. Upon code generation completion, click **Open Project** in the **Code Generation** dialog window (see *Figure 127*). This opens the project directly in the IDE.

**Figure 127. C code generation completion message**




14. In the IDE, check that heap and stack sizes are sufficient: right click the project name and select Options, then select Linker. Check **Override default** to use the icf file from STM32CubeMX generated project folder. Adjust the heap and stack sizes (see *Figure 128*).

**Figure 128. IDE workspace**




*Note: When using the MDK-ARM toolchain, go to the Application/MDK-ARM folder and double click the startup_xx.s file to edit and adjust the heap and stack sizes there.*

15. Go to the Application/User folder. Double click the main.c file and edit it.
16. The tutorial consists in creating and writing to a file on the evaluation board SD card using the FATFS file system middleware:
    a) At startup all LEDs are OFF.
    b) The red LED is turned ON and the program goes in an infinite loop if an error occurs.
    c) The orange LED is turned ON to indicate that the FATFS link has been successfully mounted on the SD driver.
    d) The blue LED is turned ON to indicate that the file has been successfully written to the SD Card.
    e) The green LED is turned ON to indicate that the file has been successfully read from file the SD Card.
17. For use case implementation, update main.c with the following user sections code snippets (see *Figure 129* and *Figure 130*).

**Figure 129. Snippets- part 1**

```
/* USER CODE BEGIN PV */
FATFS SDFatFs;  /* File system object for SD card logical drive */
FIL MyFile;     /* File object */
const char wtext[] = "Hello World!";
const uint8_t image1_bmp[] = {
0x42,0x4d,0x36,0x84,0x03,0x00,0x00,0x00,0x00,0x00,0x36,0x00,0x00,0x00,0x28,0x00,
0x00,0x00,0x40,0x01,0x00,0x00,0xf0,0x00,0x00,0x00,0x01,0x00,0x18,0x00,0x00,0x00,
0x00,0x00,0x00,0x84,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x29,0x74,0x51,0x0e,0x63,0x30,0x04,0x4c,0x1d,0x0f,
0x56,0x25,0x11,0x79,0x41,0x1f,0x85,0x6f,0x25,0x79,0x7e,0x27,0x72,0x72,0x0b,0x50,
0x43,0x00,0x44,0x15,0x00,0x4b,0x0f,0x00,0x4a,0x15,0x07,0x50,0x16,0x03,0x54,0x22,
0x23,0x70,0x65,0x30,0x82,0x6d,0x0f,0x6c,0x3e,0x22,0x80,0x5d,0x23,0x8b,0x5b,0x26};
/* USER CODE END PV */
```

```
int main(void)
{

  /* USER CODE BEGIN 1 */
  FRESULT res;                          /* FatFs function common result code */
  uint32_t byteswritten, bytesread; /* File write/read counts */
  char rtext[256];                    /* File read buffer */
  /* USER CODE END 1 */
```

**Figure 130. Snippets - part 2**

```
/* USER CODE BEGIN 3 */
  /*##-0- Turn all LEDs off(red, green, orange and blue) */
  HAL_GPIO_WritePin(GPIOG, (GPIO_PIN_10 | GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_12), GPIO_PIN_SET);
  /*##-1- FatFS: Link the SD disk I/O driver ###############################*/
  if(retSD == 0){
    HAL_GPIO_WritePin(GPIOG, GPIO_PIN_7, GPIO_PIN_RESET); /* success: set the orange LED on */
    /*##-2- Register the file system object to the FatFs module ##############*/
    if(f_mount(&SDFatFs, (TCHAR const*)SD_Path, 0) != FR_OK){
      /* FatFs Initialization Error : set the red LED on */
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
      while(1);
    }   else    {
      /*##-3- Create a FAT file system (format) on the logical drive #########*/
      /* WARNING: Formatting the uSD card will delete all content on the device */
    if(f_mkfs((TCHAR const*)SD_Path, 0, 0) != FR_OK){
      /* FatFs Format Error : set the red LED on */
      HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
      while(1);
    } else {
       /*##-4- Create and Open a new text file object with write access #####*/
      if(f_open(&MyFile, "Hello.txt", FA_CREATE_ALWAYS | FA_WRITE) != FR_OK){
        /* 'Hello.txt' file Open for write Error : set the red LED on */
        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
        while(1);
      } else {
        /*##-5- Write data to the text file ################################*/
        res = f_write(&MyFile, wtext, sizeof(wtext), (void *)&byteswritten);
        if((byteswritten == 0) || (res != FR_OK)){
          /* 'Hello.txt' file Write or EOF Error : set the red LED on */
          HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
          while(1);
        } else {
        /*##-6- Successful open/write : set the blue LED on */
          HAL_GPIO_WritePin(GPIOG, GPIO_PIN_12, GPIO_PIN_RESET);
          f_close(&MyFile);
          /*##-7- Open the text file object with read access ###############*/
          if(f_open(&MyFile, "Hello.txt", FA_READ) != FR_OK){
            /* 'Hello.txt' file Open for read Error : set the red LED on */
            HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
            while(1);
          } else {
            /*##-8- Read data from the text file ##########################*/
            res = f_read(&MyFile, rtext, sizeof(wtext), &bytesread);
            if((strcmp(rtext,wtext)!=0)|| (res != FR_OK)){
              /* 'Hello.txt' file Read or EOF Error : set the red LED on */
              HAL_GPIO_WritePin(GPIOG, GPIO_PIN_10, GPIO_PIN_RESET);
              while(1);
            } else {
              /* Successful read : set the green LED On */
              HAL_GPIO_WritePin(GPIOG, GPIO_PIN_6, GPIO_PIN_RESET);
              /*##-9- Close the open text file #############################*/
              f_close(&MyFile);
            }}}}}}}
  /*##-10- Unlink the micro SD disk I/O driver ###############################*/
  FATFS_UnLinkDriver(SD_Path);

/* Infinite loop */
while (1)
```

# 8 Tutorial 3- Getting power consumption results for a user-defined sequence

Given a microcontroller, a battery model and a user-defined power sequence, STM32CubeMX will provide an estimation of the average power consumption, battery life and average DMIPS.

## 8.1 Creating a new power sequence

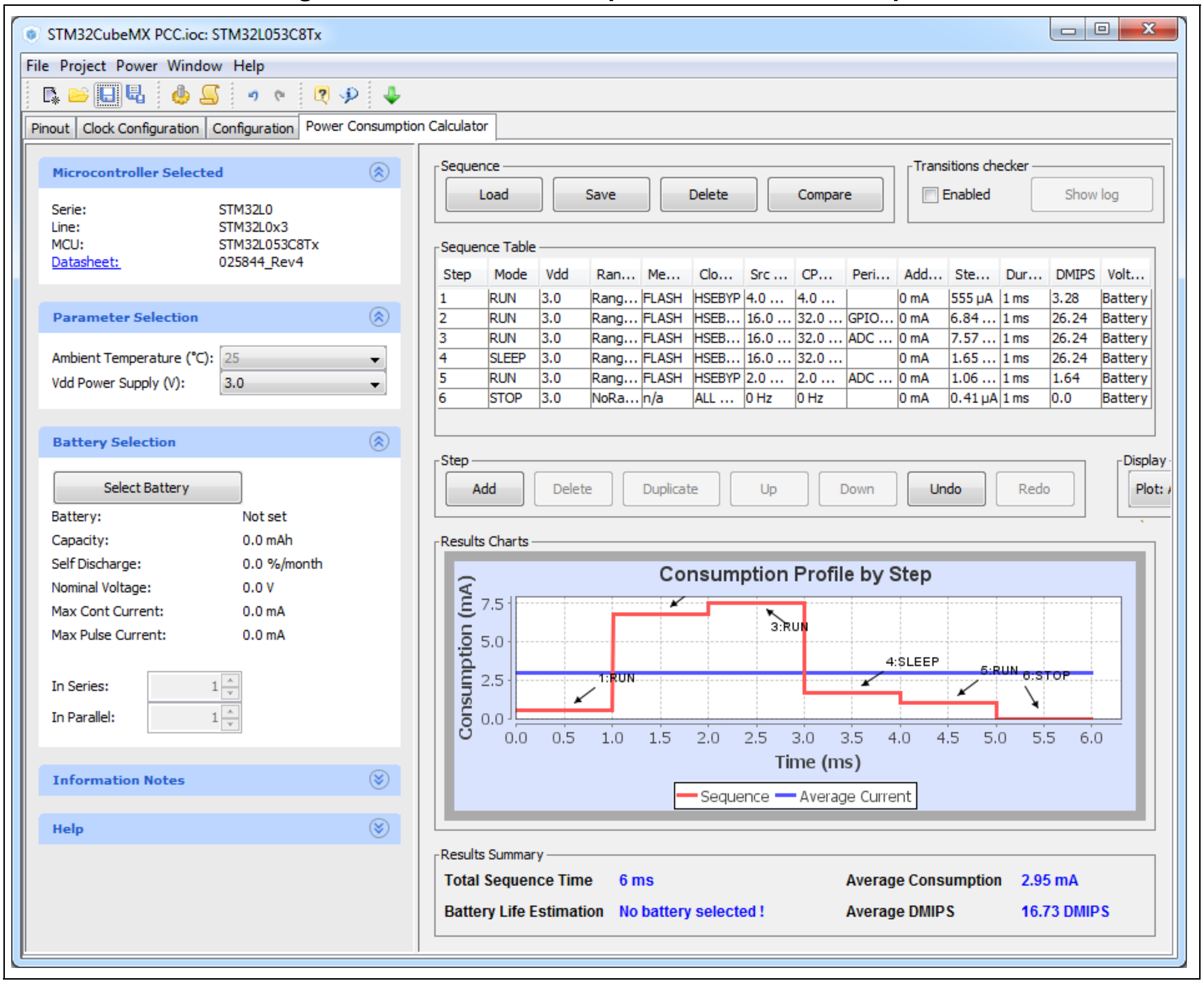
At startup, the **Sequence Table** is empty. To get results, it must be composed of at least one step.

**Sequence Table** (including step numbering), **Sequence Chart** and **Results** sections are automatically refreshed upon adding or deleting a step.

Follow the steps below to create a new sequence (see *Figure 131*):

1. Launch STM32CubeMX.
2. Click **new project** and select an MCU part number, or load an existing project.
3. Click the **Power Consumption Calculator** tab to select the Power Consumption Calculator view.
4. Select a $V_{DD}$ power supply when multiple choices are available.
5. Optionally, select a battery model to get a battery life estimate.
6. Click **Add** from the step section to open the **New step** window.

**Figure 131. Power Consumption Calculation example**




## 8.1.1 Adding a step

There are two ways to add a new step:

- Click **Add** in the **Power Consumption** panel. The **New step** window opens with empty step settings.
- Or, select a step from the sequence table and click **Duplicate**. A **New step** window opens duplicating the step settings.

## 8.1.2 Moving a step

By default, a new step is added at the end of a sequence.

Click the step in the sequence table to select it and use the **Up** and **Down** buttons to move it elsewhere in the sequence.

## 8.1.3 Deleting a step

Select the step to be deleted and click the **Delete** button.

## 8.2 Configuring a step in the power sequence

The step configuration is performed from the **Edit Step** and **New Step** windows. The graphical interface guides the user by forcing a pre-defined order for setting parameters. The parameters are set automatically by the tool when there is only one possible value (in this case, the parameter cannot be modified and is grayed out). The tool proposes only the configuration choices relevant to the selected MCU.

Proceed as follow to configure a new step:

1. Click **Add** or **Duplicate** to open the **New step** window or double-click a step from the sequence table to open the **Edit step** window.
2. Within the open step window, select in the following order:
   – The **Power Mode**

     Changing the Power Mode resets the whole step configuration.
   – The **Peripherals**

     Peripherals can be selected/unselected at any time after the Power Mode is configured.
   – The **Power scale**

     The power scale corresponds to the power consumption range (STM32L1) or the power scale (STM32F4).

     Changing the Power Mode or the Power Consumption Range discards all subsequent configurations.
   – The **Memory Fetch Type**
   – The $V_{DD}$ value if multiple choices available
   – The voltage source (battery or VBUS)
   – A **Clock Configuration**

     Changing the Clock Configuration resets the frequency choices further down.
   – When multiple choices are available, the **CPU Frequency** (STM32F4) and the **AHB Bus Frequency/CPU Frequency**(STM32L1).
3. Optionally set
   – A **step duration** (1 ms is the default value)
   – An **additional consumption** value (expressed in mA) to reflect, for example, external components used by the application (external regulator, external pull-up, LEDs or other displays). This value added to the microcontroller power consumption will impact the step overall power consumption.
4. Once the configuration is complete, the **Add** button becomes active. Click it to create the step and add it to the sequence table.

## 8.3 Reviewing results

A sequence table lists all the steps that have been defined along with their individual consumption and additional configuration parameters (see *Figure 132*).

As shown in *Figure 133*, a power **Sequence Chart** shows the average power and steps consumption in mA versus time, while the overall sequence outcomes are summarized in the **Results** section.

Choose other display options to show different results charts. For example, select pie chart to show power consumption ratios per mode (see *Figure 134*) or IP consumption charts to see the consumption per IP (see *Figure 135*).

**Figure 132. Sequence table**

Sequence Table

| Step | Mode | Vdd | Range/Scale | Memory | Clock Config | Src Freq | CPU/Bus Freq | Peripherals | Add. Current | Step Current | Duration | DMIPS | Voltage source |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | RUN | 3.0 | Range3-Low | FLASH | HSEBYP | 4.0 MHz | 4.0 MHz | | 0 mA | 555 µA | 1 ms | 3.28 | Battery |
| 2 | RUN | 3.0 | Range1-High | FLASH | HSEBYP PLL | 16.0 MHz | 32.0 MHz | GPIOA GPIOB GPIOC GPIOH | 0 mA | 6.84 mA | 1 ms | 26.24 | Battery |
| 3 | RUN | 3.0 | Range1-High | FLASH | HSEBYP PLL | 16.0 MHz | 32.0 MHz | ADC GPIOA GPIOB GPIOC GPIOH I... | 0 mA | 7.57 mA | 1 ms | 26.24 | Battery |
| 4 | SLEEP | 3.0 | Range1-High | FLASH | HSEBYP PLL | 16.0 MHz | 32.0 MHz | | 0 mA | 1.65 mA | 1 ms | 26.24 | Battery |
| 5 | RUN | 3.0 | Range3-Low | FLASH | HSEBYP | 2.0 MHz | 2.0 MHz | ADC COMP1 COMP2:Slow CRC CRS... | 0 mA | 1.06 mA | 1 ms | 1.64 | Battery |
| 6 | STOP | 3.0 | NoRange | n/a | ALL CLOCKS OFF | 0 Hz | 0 Hz | | 0 mA | 0.41 µA | 1 ms | 0.0 | Battery |

**Figure 133. Power Consumption Calculation results**

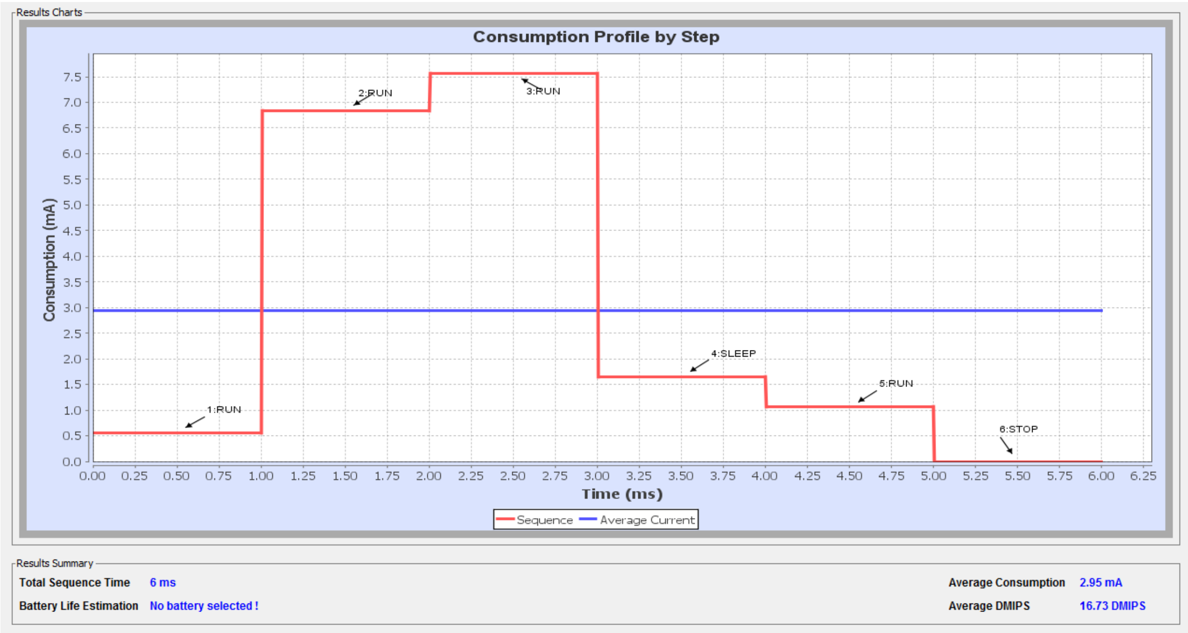


Figure 134. Power consumption results - pie chart

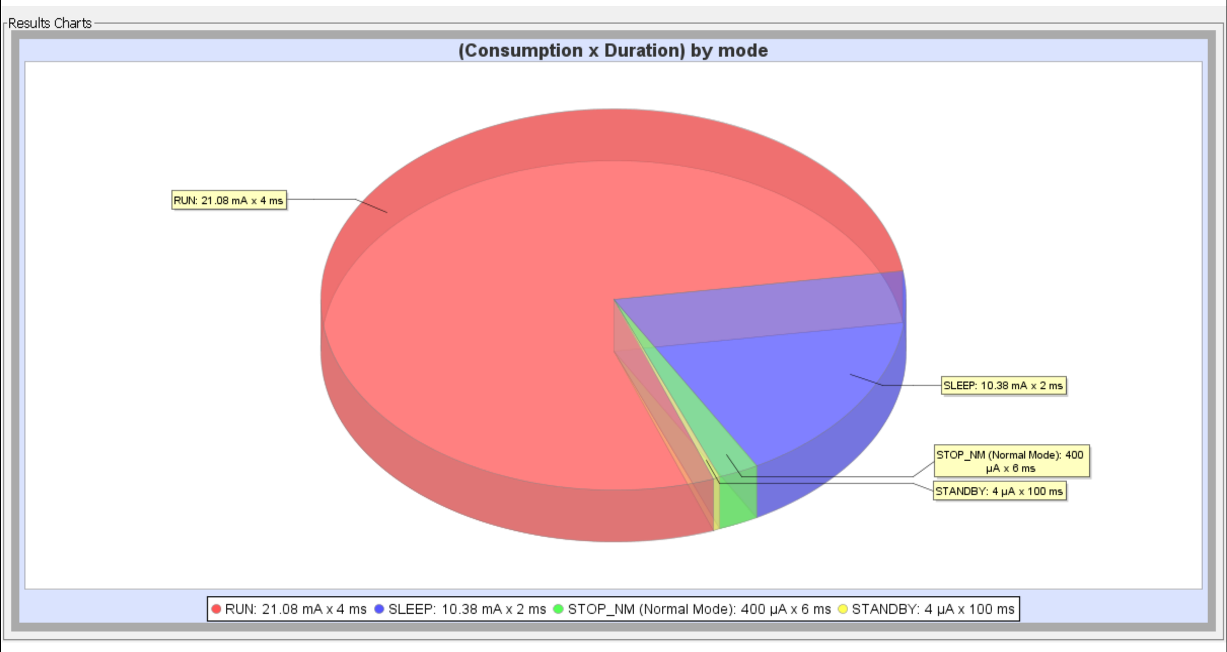
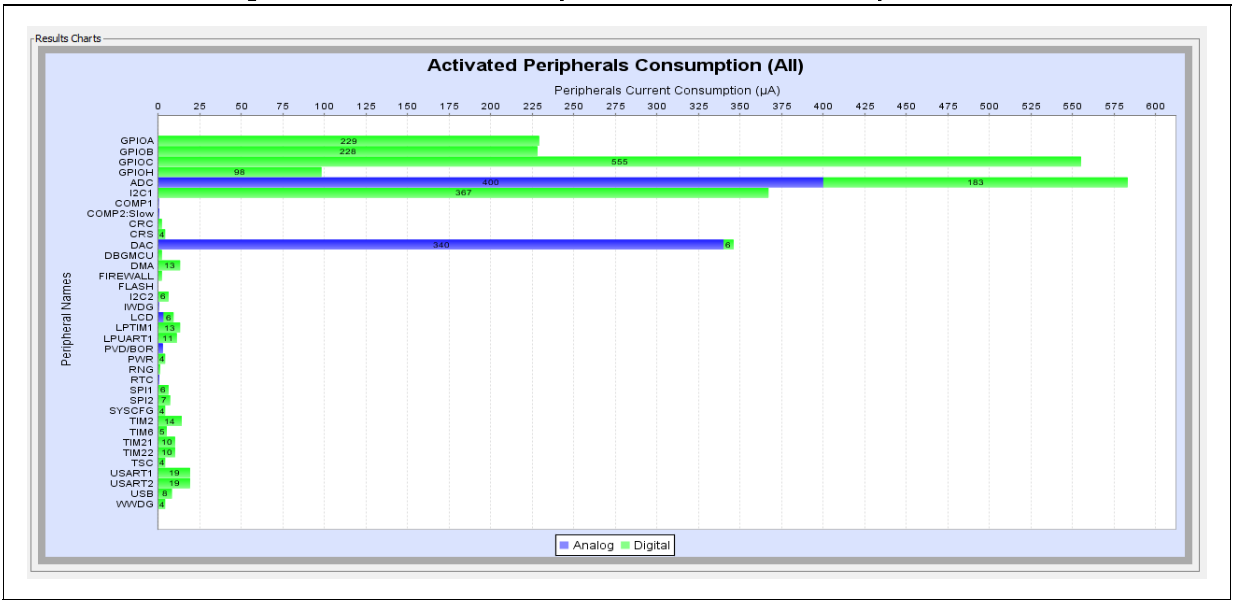



Figure 135. Power consumption results - IP consumption chart



# 9 FAQ

## 9.1 On the Pinout configuration pane, why does STM32CubeMX move some functions when I add a new peripheral mode?

You may have unselected Keep Current Signals Placement. In this case, the tool performs an automatic remapping to optimize your placement.

## 9.2 How can I manually force a function remapping?

You should use the **Manual Remapping** feature.

## 9.3 Why are some pins highlighted in yellow or in light green in the Chip view? Why cannot I change the function of some pins (when I click some pins, nothing happens)?

These pins are specific pins (such as power supply or BOOT) which are not available as peripheral signals.

## 9.4 Why do I get the error “Java 7 update 45’ when installing ‘Java 7 update 45’ or a more recent version of the JRE?

The problem generally occurs on 64-bit Windows operating system, when several versions of Java are installed on your computer and the 64-bit Java installation is too old.

During STM32CubeMX installation, the computer searches for a 64-bit installation of Java.

- If one is found, the ‘Java 7 update 45’ minimum version prerequisite is checked. If the installed version is older, an error is displayed to request the upgrade.
- If no 64-bit installation is found, STM32CubeMX searches for a 32-bit installation. If one is found and the version is too old, the ‘Java 7 update 45’ error is displayed. The user must update the installation to solve the issue.

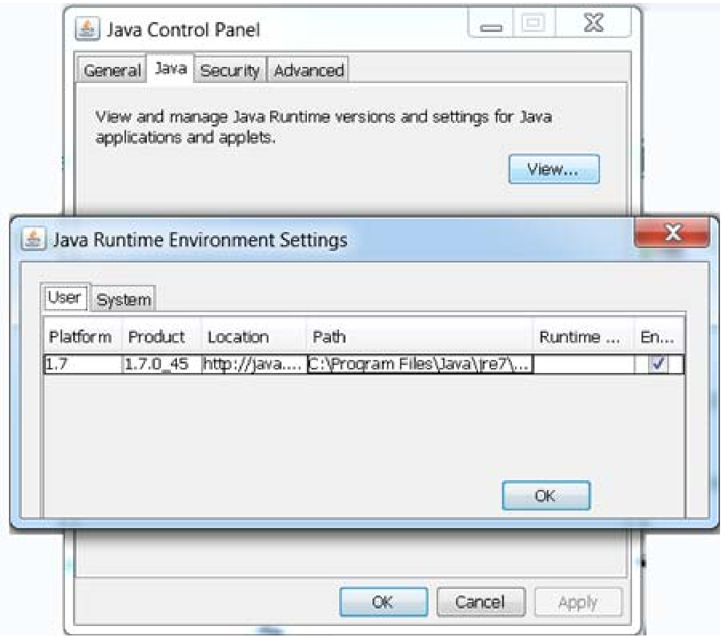
To avoid this issue from occurring, it is recommended to perform one of the following actions:

1. Remove all Java installations and reinstall only one version (32 or 64 bits) (Java 7 update 45 or more recent).
2. Keep 32-bit and 64-bit installations but make sure that the 64-bit version is at least Java 7 update 45.

*Note: Some users (Java developers for example) may need to check the PC environment variables defining hard-coded Java paths (e.g. JAVA_HOME or PATH) and update them so that they point to the latest Java installation.*

On Windows 7 you can check your Java installation using the Control Panel. To do this, double-click Java icon from Control Panel\All Control Panel to open the Java settings window (see *Figure 136*):

**Figure 136. Java Control Panel**




You can also enter *'java –version*' as an MS-DOS command to check the version of your latest Java installation (the Java program called here is a copy of the program installed under C:\Windows\System32):

```
java version "1.7.0_45"
Java (TM) SE Runtime Environment (build 1.7.0_45-b18)
Java HotSpot (TM) 64-Bit Server VM (build 24.45-b08, mixed mode)
```

## 9.5 Why does the RTC multiplexer remain inactive on the Clock tree view?

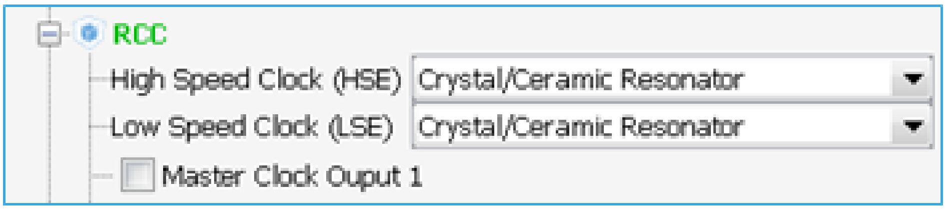
To enable the RTC multiplexer, the user shall enable the RTC IP in the **Pinout** view as indicated in below:

**Figure 137. Pinout view - Enabling the RTC**

## 9.6 How can I select LSE and HSE as clock source and change the frequency?

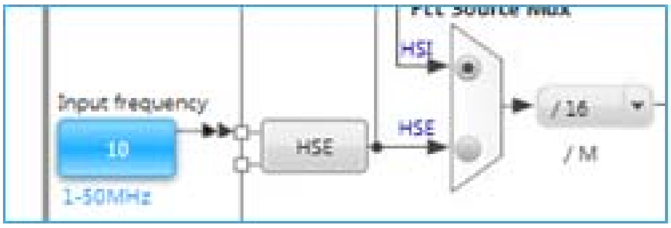
The LSE and HSE clocks become active once the RCC is configured as such in the **Pinout** view. See *Figure 138* for an example.

**Figure 138. Pinout view - Enabling LSE and HSE clocks**




The clock source frequency can then be edited and the external source selected:

**Figure 139. Pinout view - Setting LSE/HSE clock frequency**




## 9.7 Why STM32CubeMX does not allow me to configure PC13, PC14, PC15 and PI8 as outputs when one of them is already configured as an output?

STM32CubeMX implements the restriction documented in the reference manuals as a footnote in table Output Voltage characteristics:

*"PC13, PC14, PC15 and PI8 are supplied through the power switch. Since the switch only sinks a limited amount of current (3 mA), the use of GPIOs PC13 to PC15 and PI8 in output mode is limited: the speed should not exceed 2 MHz with a maximum load of 30 pF and these I/Os must not be used as a current source (e.g. to drive a LED)."*

# Appendix A STM32CubeMX pin assignment rules

The following pin assignment rules are implemented in STM32CubeMX:

- Rule 1: Block consistency
- Rule 2: Block inter-dependency
- Rule 3: One block = one peripheral mode
- Rule 4: Block remapping (only for STM32F10x)
- Rule 5: Function remapping
- Rule 6: Block shifting (only for STM32F10x)
- Rule 7: Setting or clearing a peripheral mode
- Rule 8: Mapping a function individually (if Keep Current Placement is unchecked)
- Rule 9: GPIO signals mapping

## A.1 Block consistency

When setting a pin signal (provided there is no ambiguity about the corresponding peripheral mode), all the pins/signals required for this mode are mapped and pins are shown in green (otherwise the configured pin is shown in orange).

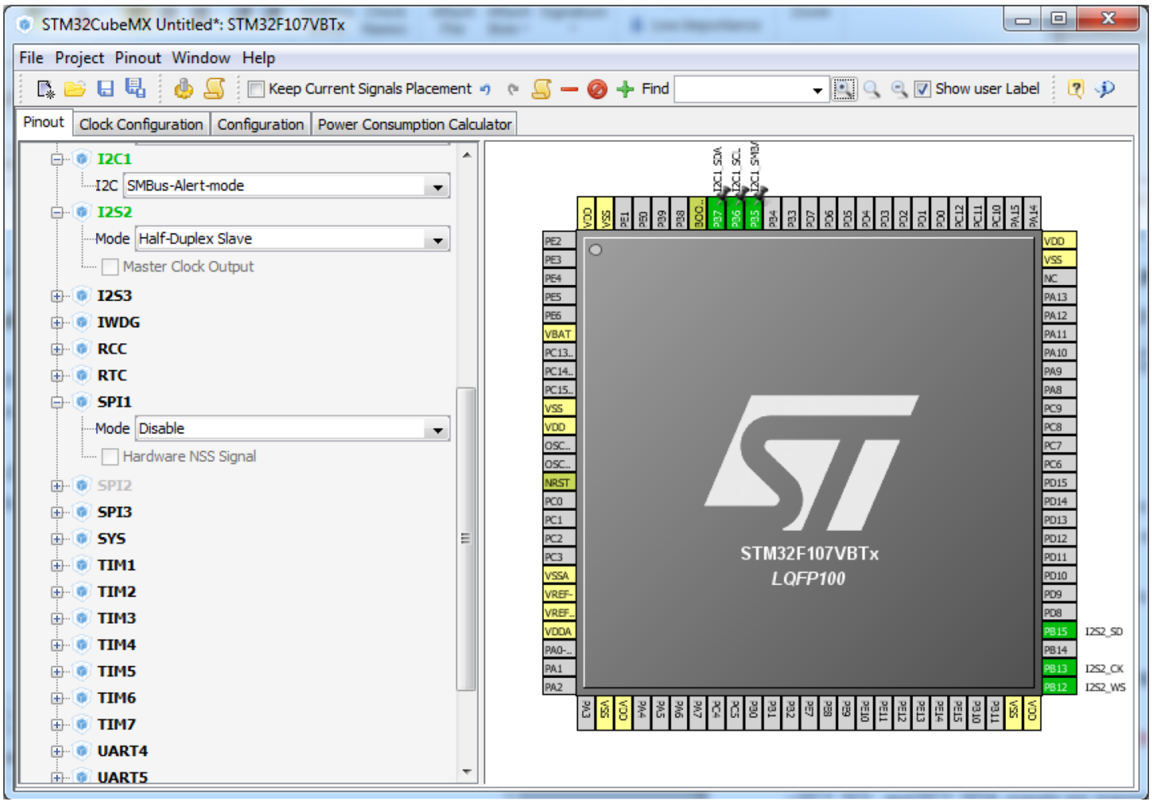
When clearing a pin signal, all the pins/signals required for this mode are unmapped simultaneously and the pins turn back to gray.

### Example of block mapping with a STM32F107x MCU

If the user assigns I2C1_SMBA function to PB5, then STM32CubeMX configures pins and modes as follows:

- I2C1_SCL and I2C1_SDA signals are mapped to the PB6 and PB7 pins, respectively (see *Figure 140*).
- I2C1 peripheral mode is set to SMBus-Alert mode.

**Figure 140. Block mapping**

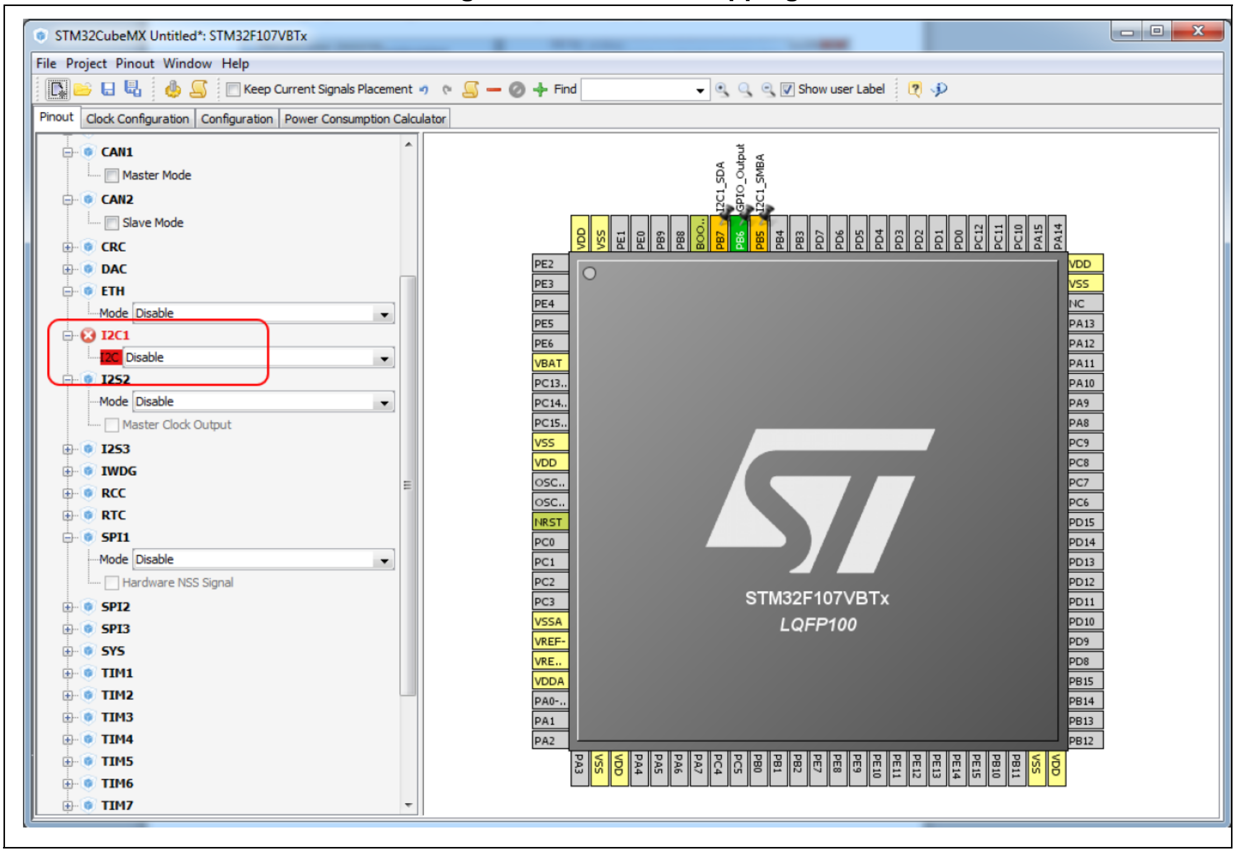



## Example of block remapping with a STM32F107x MCU

If the user assigns GPIO_Output to PB6, STM32CubeMX automatically disables I2C1 SMBus-Alert peripheral mode from the peripheral tree view and updates the other I2C1 pins (PB5 and PB7) as follows:

- If they are unpinned, the pin configuration is reset (pin grayed out).
- If they are pinned, the peripheral signal assigned to the pins is kept and the pins are highlighted in orange since they no longer match a peripheral mode (see *Figure 141*).

**Figure 141. Block remapping**




For STM32CubeMX to find an alternative solution for the I2C peripheral mode, the user will need to unpin I2C1 pins and select the I2C1 mode from the peripheral tree view (see *Figure 142* and *Figure 143*).

Figure 142. Block remapping - example 1

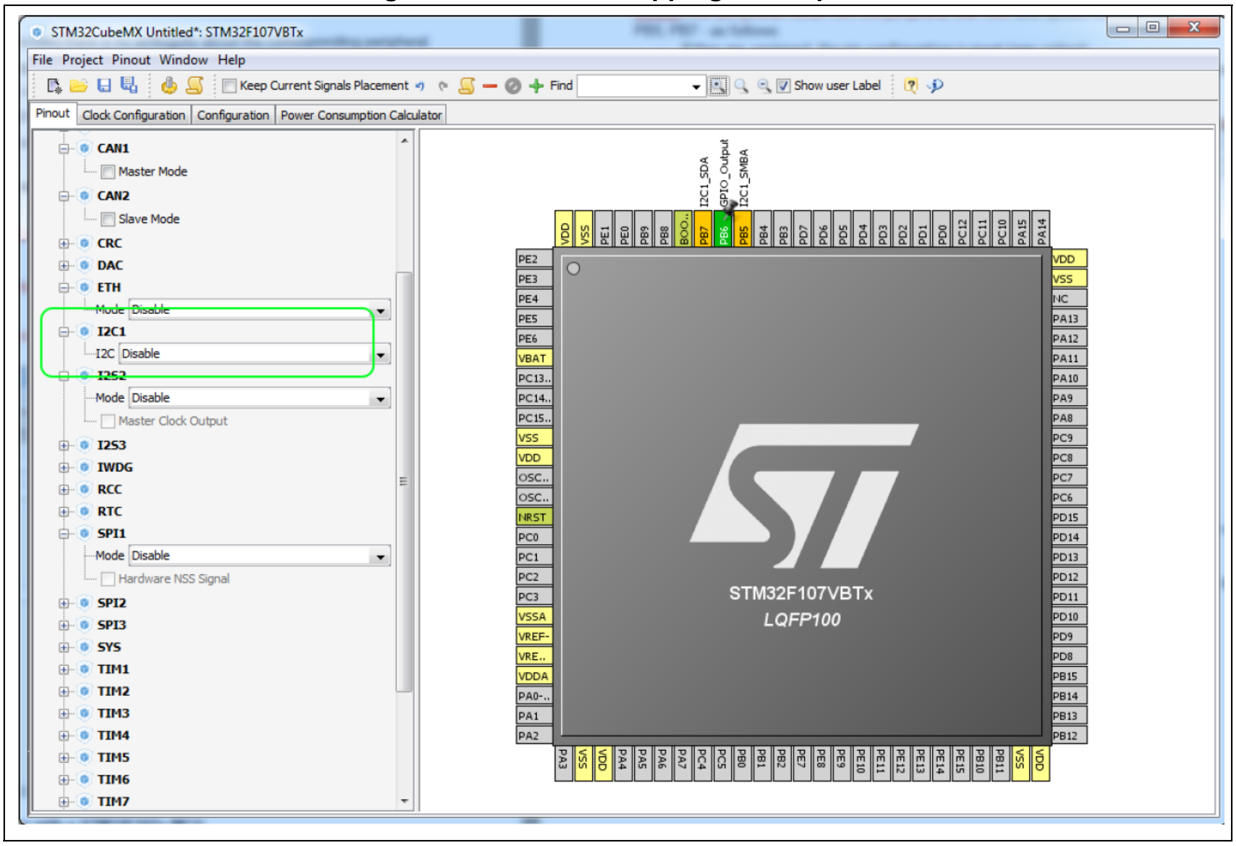
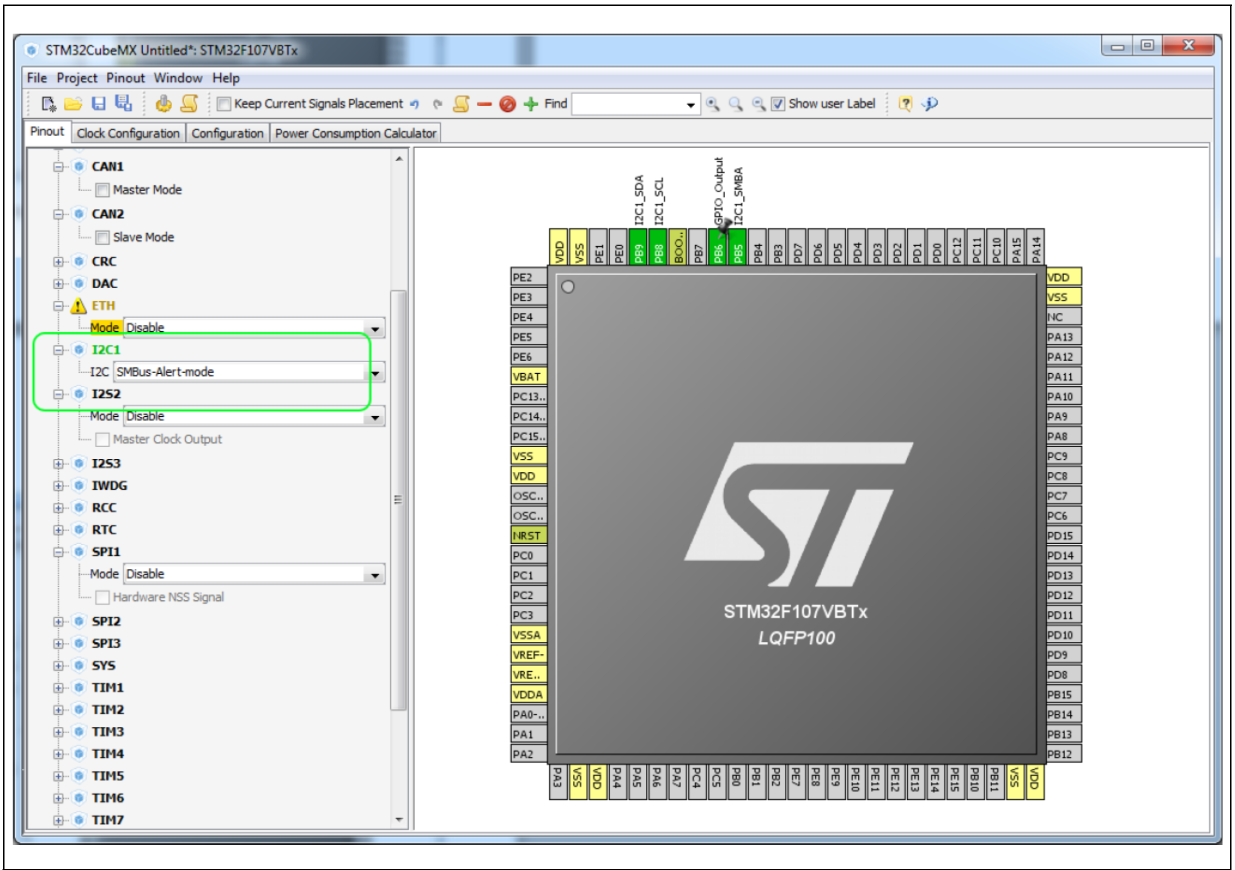


**Figure 143. Block remapping - example 2**



## A.2 Block inter-dependency

On the **Chip** view, the same signal can appear as an alternate function for multiple pins. However it can be mapped only once.

As a consequence, for STM32F1 MCUs, two blocks of pins cannot be selected simultaneously for the same peripheral mode: when a block/signal from a block is selected, the alternate blocks are cleared.

### Example of block remapping of SPI in full-duplex master mode with a STM32F107x MCU

If SPI1 full-duplex master mode is selected from the tree view, by default the corresponding SPI signals are assigned to PB3, PB4 and PB5 pins (see *Figure 144*).

If the user assigns to PA6 the SPI1_MISO function currently assigned to PB4, STM32CubeMX clears the PB4 pin from the SPI1_MISO function, as well as all the other pins configured for this block, and moves the corresponding SPI1 functions to the relevant pins in the same block as the PB4 pin (see *Figure 145*).

(by pressing CTRL and clicking PB4 to show PA6 alternate function in blue, then drag and drop the signal to pin PA6)

**Figure 144. Block inter-dependency - SPI signals assigned to PB3/4/5**

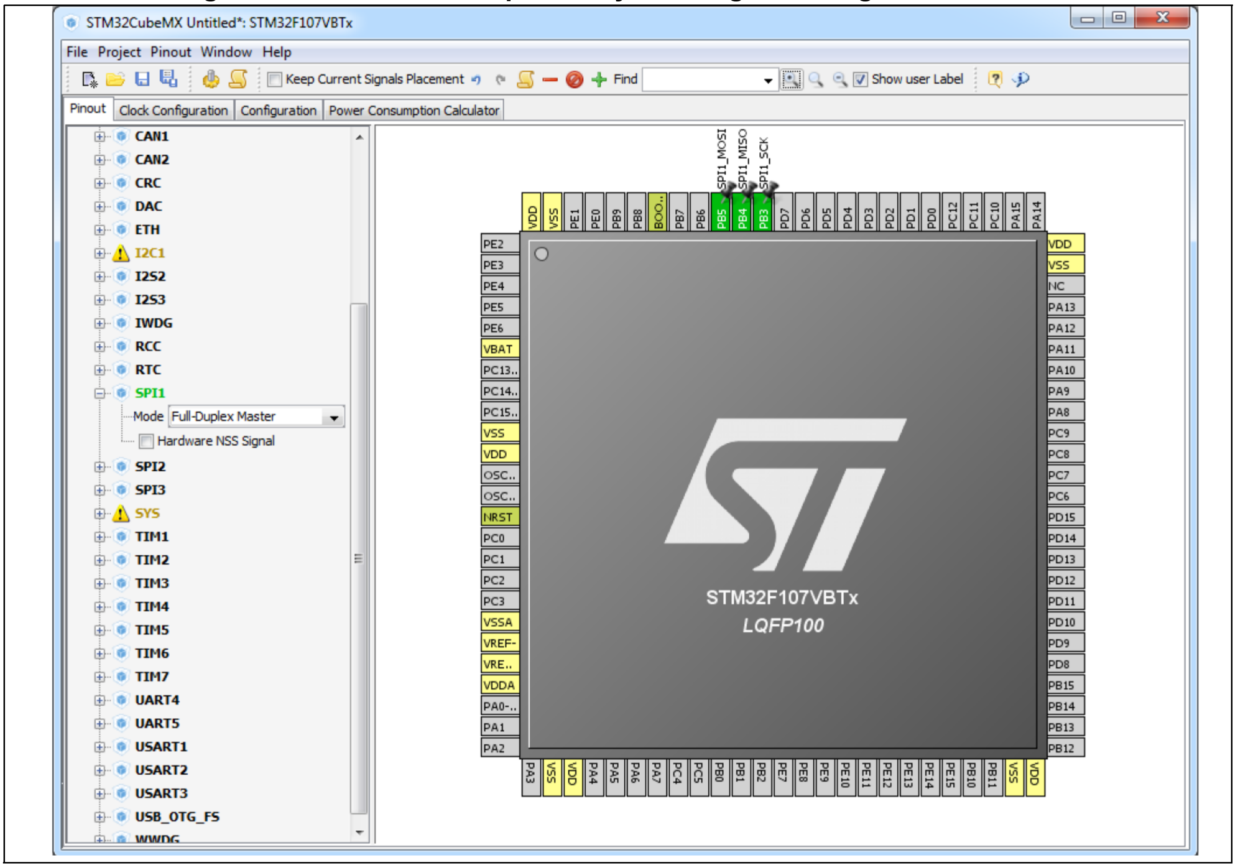
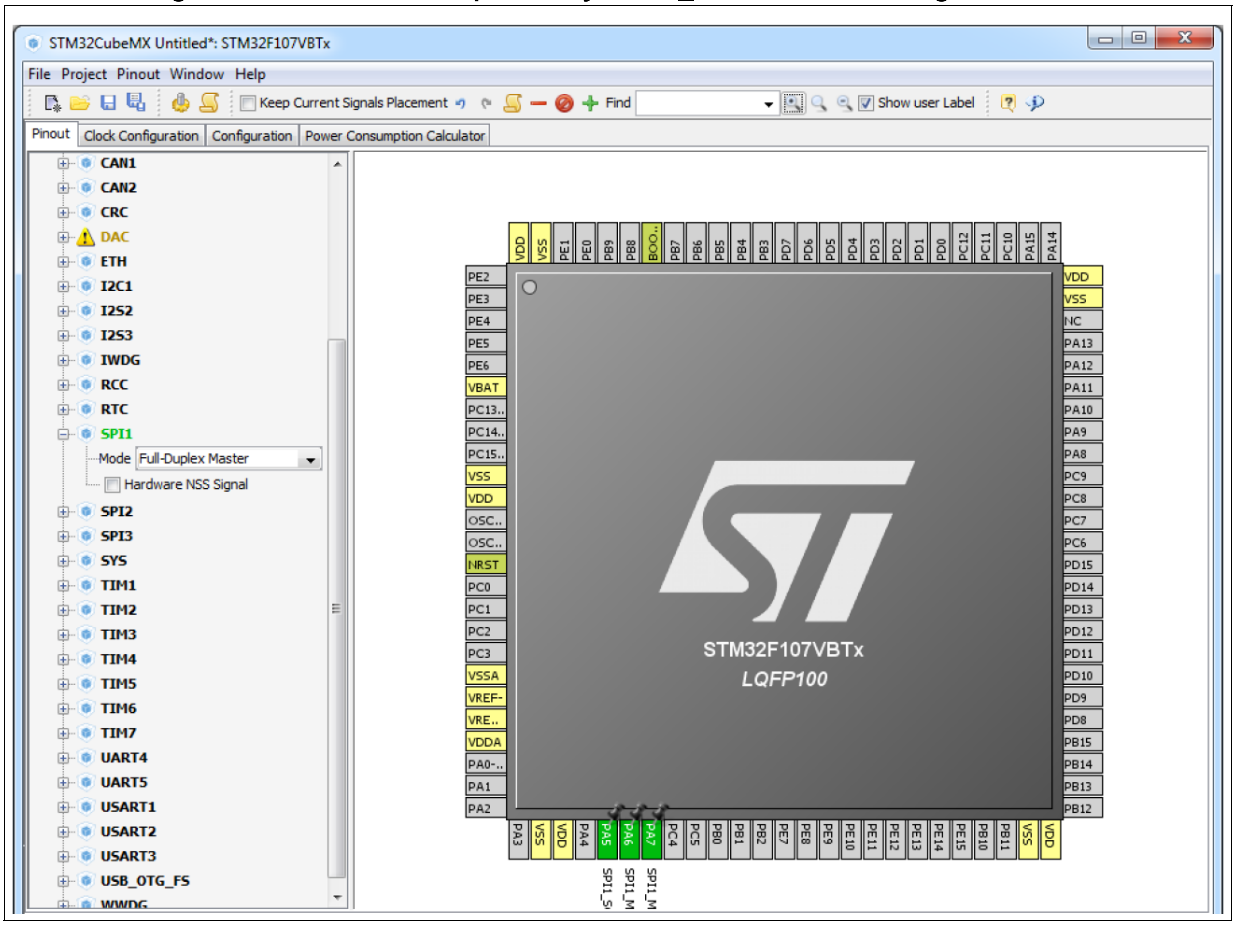


**Figure 145. Block inter-dependency - SPI1_MISO function assigned to PA6**



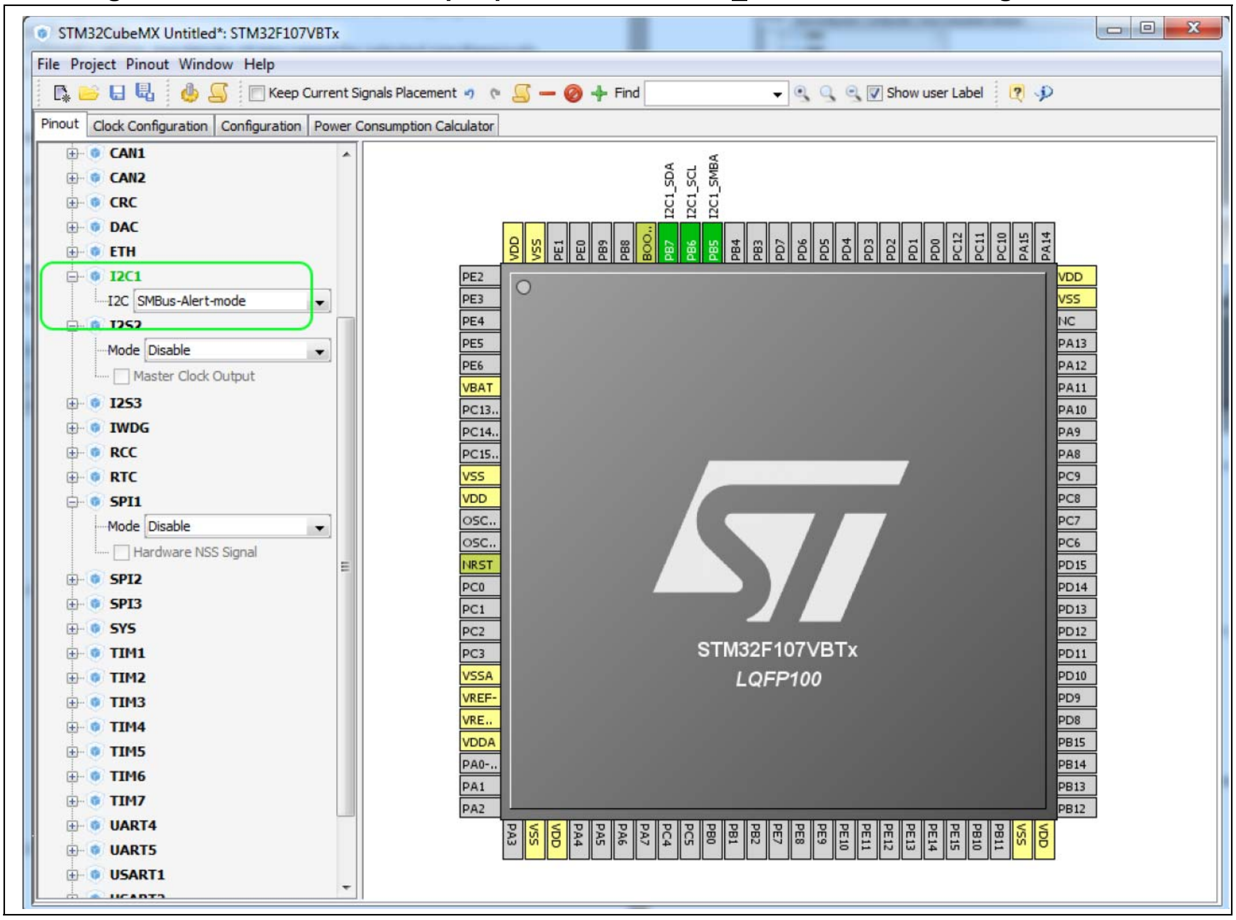
## A.3 One block = one peripheral mode

When a block of pins is fully configured in the **Chip** view (shown in green), the related peripheral mode is automatically set in the Peripherals tree.

### Example of STM32F107x MCU

Assigning the I2C1_SMBA function to PB5 automatically configures I2C1 peripheral in SMBus-Alert mode (see Peripheral tree in *Figure 146*).

**Figure 146. One block = one peripheral mode - I2C1_SMBA function assigned to PB5**




## A.4 Block remapping (STM32F10x only)

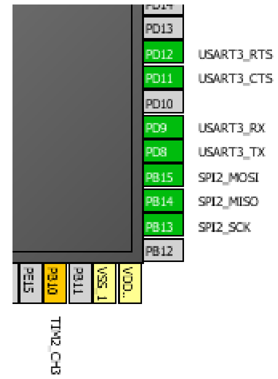
To configure a peripheral mode, STM32CubeMX selects a block of pins and assigns each mode signal to a pin in this block. In doing so, it looks for the first free block to which the mode can be mapped.

When setting a peripheral mode, if at least one pin in the default block is already used, STM32CubeMX tries to find an alternate block. If none can be found, it either selects the functions in a different sequence, or unchecks Keep Current Signals Placement, and remaps all the blocks to find a solution.

### Example

STM32CubeMX remaps USART3 hardware-flow-control mode to the (PD8-PD9-PD11-PD12) block, because PB14 of USART3 default block is already allocated to the SPI2_MISO function (see *Figure 147*).

**Figure 147. Block remapping - example 2**



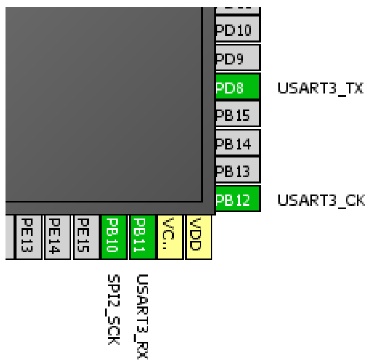

## A.5 Function remapping

To configure a peripheral mode, STM32CubeMX assigns each signal of the mode to a pin. In doing so, it will look for the first free pin the signal can be mapped to.

### Example using STM32F415x

When configuring USART3 for the Synchronous mode, STM32CubeMX discovered that the default PB10 pin for USART3_TX signal was already used by SPI. It thus remapped it to PD8 (see *Figure 148*).

**Figure 148. Function remapping example**



## A.6 Block shifting (only for STM32F10x and when “Keep Current Signals placement” is unchecked)

If a block cannot be mapped and there are no free alternate solutions, STM32CubeMX tries to free the pins by remapping all the peripheral modes impacted by the shared pin.

### Example

With the Keep current signal placement enabled, if USART3 synchronous mode is set first, the Asynchronous default block (PB10-PB11) is mapped and Ethernet becomes unavailable (shown in red) (see *Figure 149*).

Unchecking ☐ Keep Current Signals Placement allows STM32CubeMX shifting blocks around and freeing a block for the Ethernet MII mode. (see *Figure 150*).

**Figure 149. Block shifting not applied**

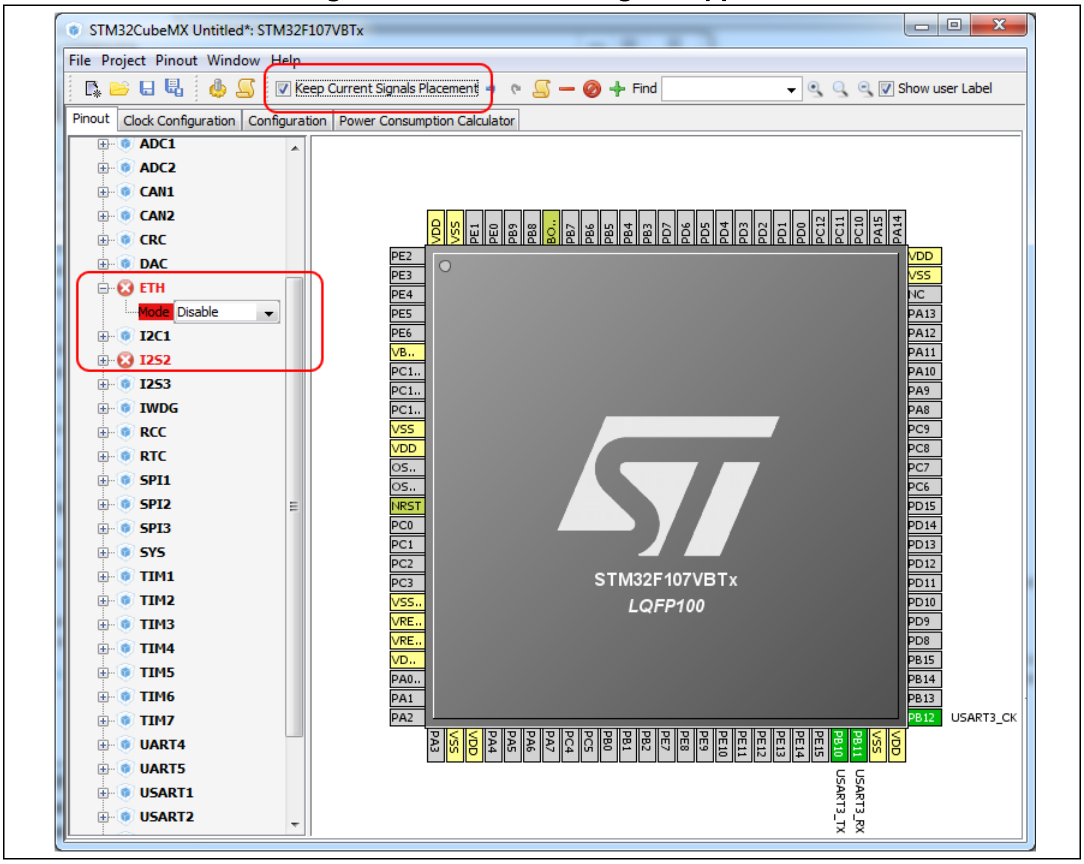
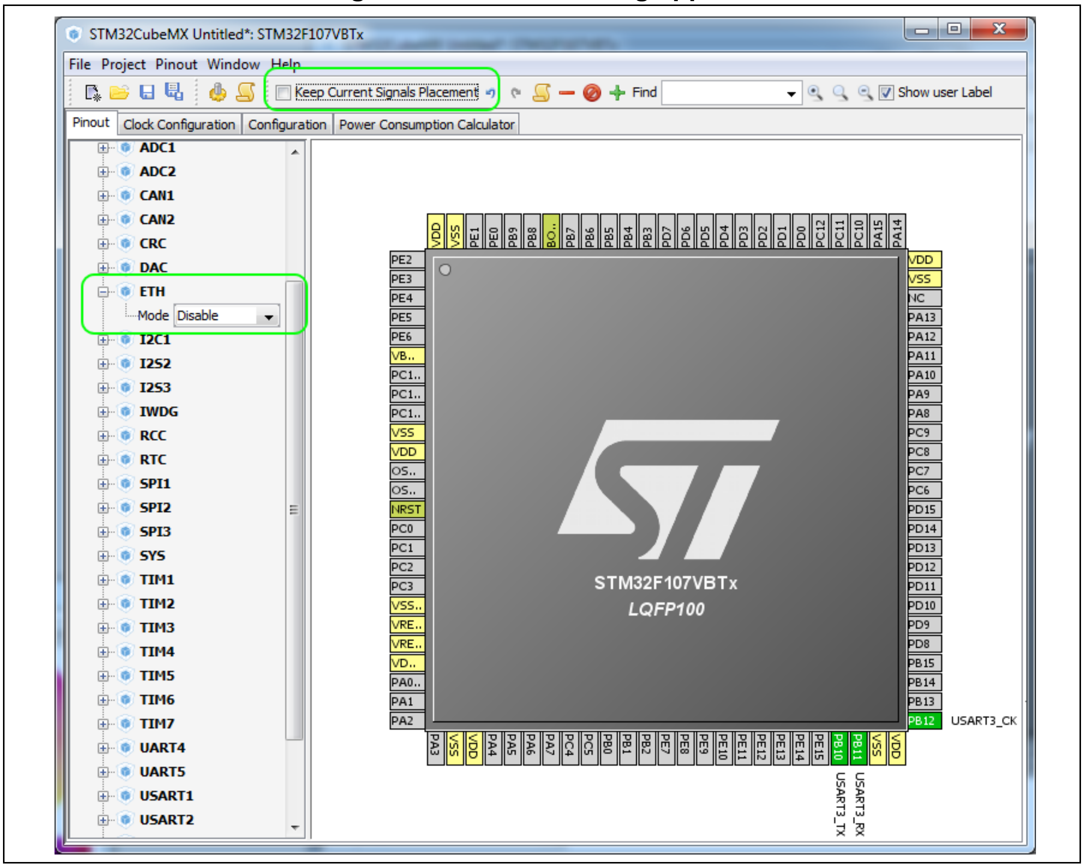


**Figure 150. Block shifting applied**




## A.7 Setting and clearing a peripheral mode

The Peripherals panel and the **Chip** view are linked: when a peripheral mode is set or cleared, the corresponding pin functions are set or cleared.

## A.8 Mapping a function individually

When STM32CubeMX needs a pin that has already been assigned manually to a function (no peripheral mode set), it can move this function to another pin, only if ☐ Keep Current Signals Placement is unchecked and the function is not pinned (no pin icon).

## A.9 GPIO signals mapping

I/O signals (GPIO_Input, GPIO_Output, GPIO_Analog) can be assigned to pins either manually through the **Chip** view or automatically through the Pinout menu. Such pins can no longer be assigned automatically to another signal: STM32CubeMX signal automatic placement does not take into account this pin anymore since it does not shift I/O signals to other pins.

The pin can still be manually assigned to another signal or to a reset state.

# Appendix B STM32CubeMX C code generation design choices and limitations

This section summarizes STM32CubeMX design choices and limitations.

## B.1 STM32CubeMX generated C code and user sections

The C code generated by STM32CubeMX provides user sections as illustrated below. They allow user C code to be inserted and preserved at next C code generation.

User sections shall neither be moved nor renamed. Only the user sections defined by STM32CubeMX are preserved. User created sections will be ignored and lost at next C code generation.

```
 /* USER CODE BEGIN 0 */
(..)
/* USER CODE END 0 */
```

*Note: STM32CubeMX may generate C code in some user sections. It will be up to the user to clean the parts that may become obsolete in this section. For example, the while(1) loop in the main function is placed inside a user section as illustrated below:*

```
/* USER CODE BEGIN 3 */
  /* Infinite loop */
  while (1)
  {

  }
/* USER CODE END 3 */
```

## B.2 STM32CubeMX design choices for peripheral initialization

STM32CubeMX generates peripheral *_Init* functions that can be easily identified thanks to the MX_ prefix:

```
static void MX_GPIO_Init(void);
static void MX_<Peripheral Instance Name>_Init(void);
static void MX_I2S2_Init(void);
```

An *MX_<peripheral instance name>_Init* function exists for each peripheral instance selected by the user (e.g, *MX_I2S2_Init*). It performs the initialization of the relevant handle structure (e.g, &hi2s2 for I2S second instance) that is required for HAL driver initialization (e.g., *HAL_I2S_Init*) and the actual call to this function:

```
void MX_I2S2_Init(void)
{
 hi2s2.Instance = SPI2;
  hi2s2.Init.Mode = I2S_MODE_MASTER_TX;
  hi2s2.Init.Standard = I2S_STANDARD_PHILLIPS;
  hi2s2.Init.DataFormat = I2S_DATAFORMAT_16B;
  hi2s2.Init.MCLKOutput = I2S_MCLKOUTPUT_DISABLE;
```

```
  hi2s2.Init.AudioFreq = I2S_AUDIOFREQ_192K;
  hi2s2.Init.CPOL = I2S_CPOL_LOW;
  hi2s2.Init.ClockSource = I2S_CLOCK_PLL;
  hi2s2.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_ENABLE;
  HAL_I2S_Init(&hi2s2);
}
```

By default, the peripheral initialization is done in *main.c*. If the peripheral is used by a middleware mode, the peripheral initialization can be done in the middleware corresponding .c file.

Customized *HAL_<IP Name>_MspInit()* functions are created in the stm32f4xx_hal_msp.c file to configure the low level hardware (GPIO, CLOCK) for the selected IPs.

## B.3 STM32CubeMX design choices and limitations for middleware initialization

### B.3.1 Overview

STM32CubeMX generates middleware *Init* functions that can be easily identified thanks to the MX_ prefix:

```
MX_LWIP_Init(); // defined in lwip.h file
MX_USB_HOST_Init(); // defined in usb_host.h file
MX_FATFS_Init(); // defined in fatfs.h file
```

Note however the following exceptions:

- No *Init* function is generated for FreeRTOS unless the user chooses, from the Project settings window, to generate *Init* functions as pairs of .c/.h files. Instead, a *StartDefaultTask* function is defined in the *main.c* file and CMSIS-RTOS native function (*osKernelStart*) is called in the main function.
- If FreeRTOS is enabled, the *Init* functions for the other middlewares in use are called from the *StartDefaultTask* function in the main.c file.

  Example:

```
void StartDefaultTask(void const * argument)
{
   /* init code for FATFS */
   MX_FATFS_Init();
   /* init code for LWIP */
   MX_LWIP_Init();
  /* init code for USB_HOST */
  MX_USB_HOST_Init();
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1);
  }
  /* USER CODE END 5 */
```

```
}
```

### B.3.2 USB Host

USB peripheral initialization is performed within the middleware initialization C code in the *usbh_conf.c* file, while USB stack initialization is done within the *usb_host.c* file.

When using the USB Host middleware, the user is responsible for implementing the *USBH_UserProcess* callback function in the generated *usb_host.c* file.

From STM32CubeMX user interface, the user can select to register one class or all classes if the application requires switching dynamically between classes.

### B.3.3 USB Device

USB peripheral initialization is performed within the middleware initialization C code in the *usbd_conf.c* file, while USB stack initialization is done within the *usb_device.*c file.

USB VID, PID and String standard descriptors are configured via STM32CubeMX user interface and available in the *usbd_desc.c* generated file. Other standard descriptors (configuration, interface) are hard-coded in the same file preventing support for USB composite devices.

When using the USB Device middleware, the user is responsible for implementing the functions in the *usbd_<classname>_if.c* class interface file for all device classes (e.g., usbd_storage_if.c).

USB MTP and CCID classes are not supported.

### B.3.4 FATFS

FATFS configuration is available in the *fatfs_handles.h* and *ffconf.h* generated files.

The initialization of the SDIO peripheral for the FATFS SD Card mode and of the FMC peripheral for the FATFS External SDRAM and External SRAM modes are kept in the *main.c* file.

Some files need to be modified by the user to match user board specificities (BSP drivers in STM32Cube embedded software package can be used as example):

- *bsp_driver_sd.c/.h* generated files when using FATFS SD Card mode
- *bsp_driver_sram.c/.h* generated files when using FATFS External SRAM mode
- *bsp_driver_sdram.c/.h* generated files when using FATFS External SDRAM mode.

Multi-drive FATFS is supported, which means that multiple logical drives can be used by the application (External SDRAM, External SRAM, SD Card, USB Disk, User defined). However support for multiple instances of a given logical drive is not available (e.g. FATFS using two instances of USB hosts or several RAM disks).

NOR and NAND Flash memory are not supported. In this case, the user shall select the FATFS user-defined mode and update the *user_diskio.c* driver file generated to implement the interface between the middleware and the selected peripheral.

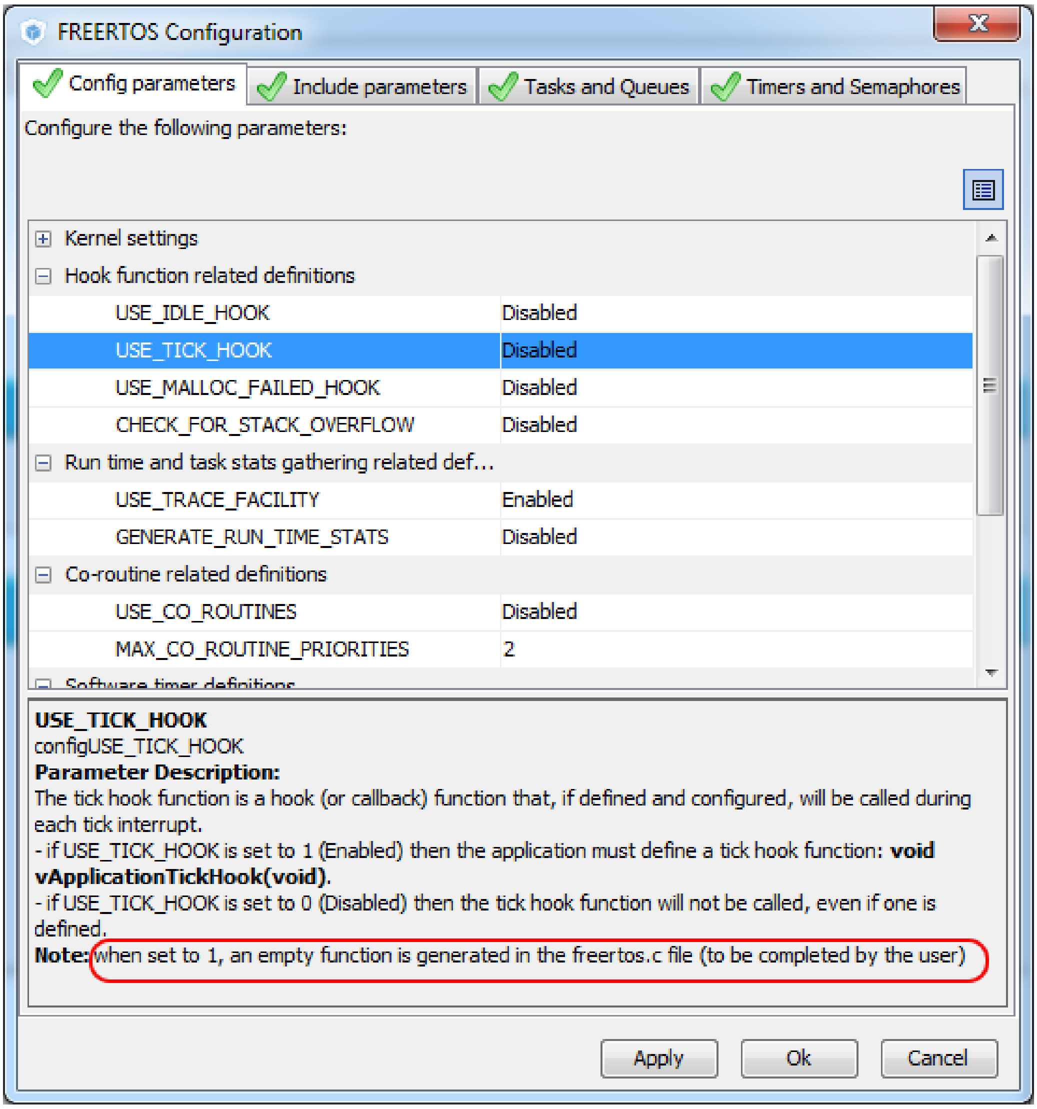
### B.3.5 FreeRTOS

FreeRTOS configuration is available in *FreeRTOSConfig.h* generated file.

When FreeRTOS is enabled, all other selected middleware modes (e.g., LwIP, FATFS, USB) will be initialized within the same FreeRTOS thread in the main.c file.

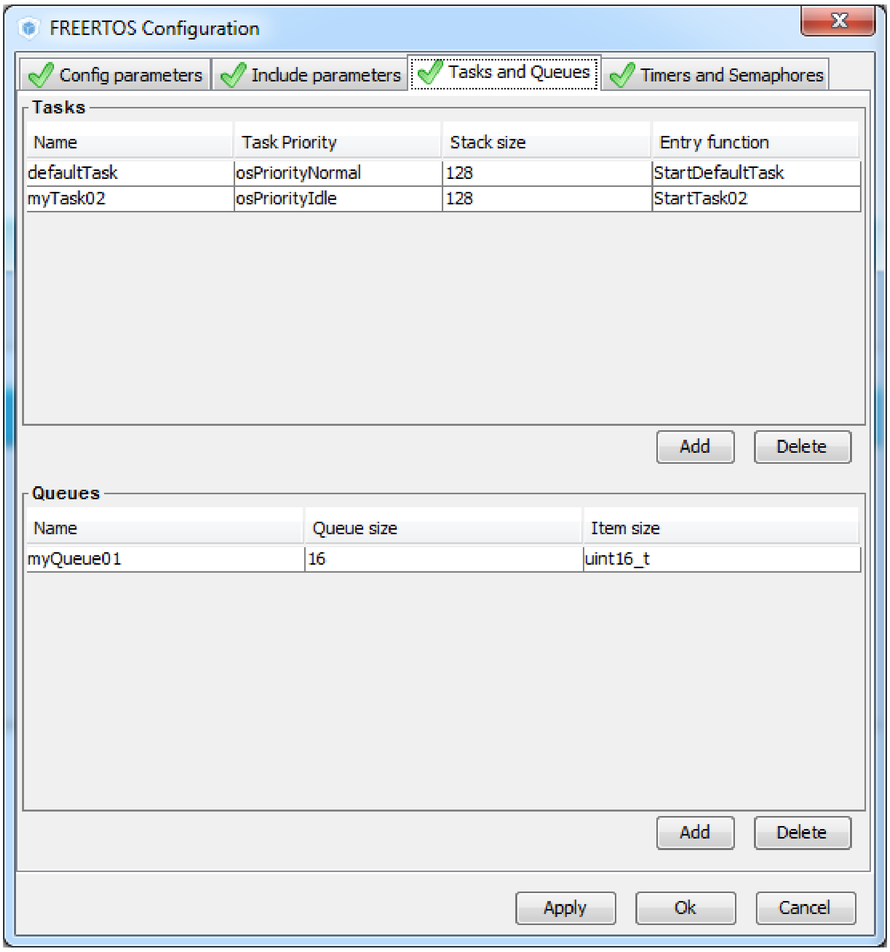
When GENERATE_RUN_TIME_STATS, CHECK_FOR_STACK_OVERFLOW, USE_IDLE_HOOK, USE_TICK_HOOK and USE_MALLOC_FAILED_HOOK parameters are activated, STM32CubeMX generates *freertos.c* file with empty functions that the user shall implement. This is highlighted by the tooltip (see *Figure 151*).

**Figure 151. FreeRTOS HOOK functions to be completed by user**




Through STM32CubeMX FreeRTOS configuration window, the user can configure all the resources required for the real-time OS application: tasks, queues, semaphores and timers. The corresponding freeRTOS elements will be defined and created in the generated code (see *Figure 152*).

**Figure 152. FreeRTOS elements**




## B.3.6 LwIP

LwIP initialization function is defined in *lwip.c*, while LwIP configuration is available in *lwipopts.h* generated file.

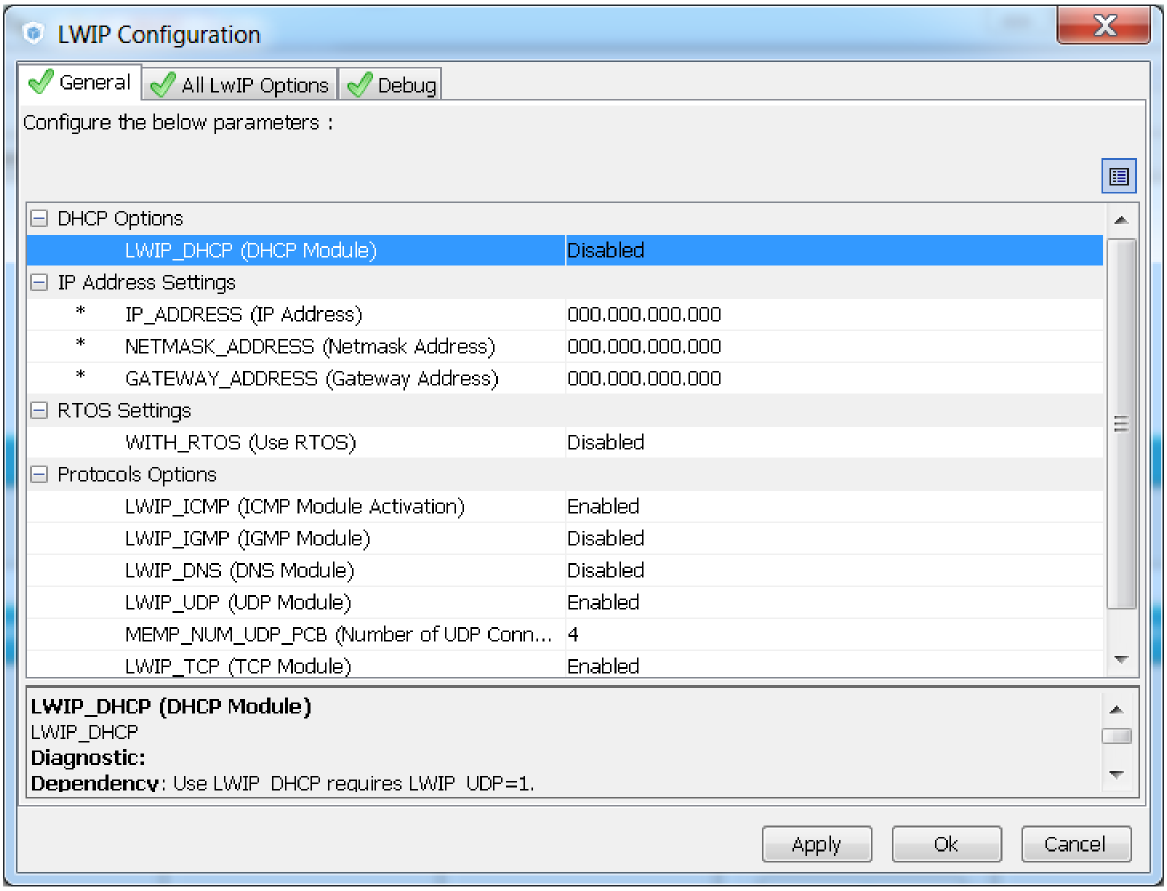
STM32CubeMX supports LwIP over Ethernet only. The Ethernet peripheral initialization is done within the middleware initialization C code.

STM32CubeMX does not support user C code insertion in stack native files. However, some LwIP use cases require modifying stack native files (e.g., *cc.h, mib2.c*): user modifications shall be backed up since they will be lost at next STM32CubeMX generation.

STM32CubeMX LwIP configuration does not support IPv6.

DHCP must be disabled, to configure a static IP address (see *Figure 153*).

**Figure 153. LwIP configuration**




STM32CubeMX generated C code will report compilation errors when specific parameters are enabled (disabled by default). The user must fix the issues with a stack patch (downloaded from Internet) or user C code. The following parameters generate an error:

- MEM_USE_POOLS: user C code to be added either in *lwipopts.h* or in *cc.h* (stack file).
- PPP_SUPPORT, PPPOE_SUPPORT: user C code required
- MEMP_SEPARATE_POOLS with MEMP_OVERFLOW_CHECK > 0: a stack patch required
- MEM_LIBC_MALLOC & RTOS enabled: stack patch required
- LWIP_EVENT_API: stack patch required

In STM32CubeMX, the user must enable FreeRTOS in order to use LwIP with the netconn and sockets APIs. These APIs require the use of threads and consequently of an operating system. Without FreeRTOS, only the LwIP event-driven raw API can be used.

# Appendix C STM32 microcontrollers naming conventions

STM32 microcontroller part numbers are codified following the below naming conventions:

- Device subfamilies

  The higher the number, the more features available.

  For example STM32L0 line includes STM32L051, L052, L053, L061, L062, L063 subfamilies where STM32L06x part numbers come with AES while STM32L05x do not.

  The last digit indicates the level of features. In the above example:
  - 1 =Access line
  - 2 = with USB
  - 3 = with USB and LCD.
- Pin counts
  - F = 20 pins
  - G = 28 pins
  - K = 32 pins
  - T = 36 pins
  - S = 44 pins
  - C = 48 pins
  - R = 64 pins (or 66 pins)
  - M = 80 pins
  - O = 90 pins
  - V = 100 pins
  - Q= 132 pins (e. g. STM32L162QDH6)
  - Z=144
  - I=176 (+25)
  - B = 208 pins (e. g.: STM32F429BIT6)
  - N = 216 pins
- Flash memory sizes
  - 4 = 16 Kbytes of Flash memory
  - 6 = 32 Kbytes of Flash memory
  - 8 = 64 Kbytes of Flash memory
  - B = 128 Kbytes of Flash memory
  - C = 256 Kbytes of Flash memory
  - D = 384 Kbytes of Flash memory
  - E = 512 Kbytes of Flash memory
  - F = 768 Kbytes of Flash memory
  - G = 1024 Kbytes of Flash memory
  - I = 2048 Kbytes of Flash memory
- Packages
  - B = SDIP
  - H = BGA

- M = SO
- P = TSSOP
- T = LQFP
- U = VFQFPN
- Y = WLCSP

*Figure 154* shows an example of STM32 microcontroller part numbering scheme.

**Figure 154. STM32 microcontroller part numbering scheme**

Example: STM32 F 439 V I T 6 xxx

**Device family**

STM32 = ARM-based 32-bit microcontroller

**Product type**

F = general-purpose

**Device subfamily**

437= STM32F437xx, USB OTG FS/HS, camera interface, Ethernet, cryptographic acceleration

439= STM32F439xx, USB OTG FS/HS, camera interface, Ethernet, LCD-TFT, cryptographic acceleration

**Pin count**

V = 100 pins

Z = 144 pins

A = 169 pins

I = 176 pins

B = 208 pins

N = 216 pins

**Flash memory size**

G = 1024 Kbytes of Flash memory

I = 2048 Kbytes of Flash memory

**Package**

T = LQFP

H = BGA

Y = WLCSP

**Temperature range**

6 = Industrial temperature range, –40 to 85 °C.

7 = Industrial temperature range, –40 to 105 °C.

**Options**

xxx = programmed parts

TR = tape and reel

# Appendix D STM32 microcontrollers power consumption parameters

This section provides an overview on how to use STM32CubeMX Power Consumption Calculator (PCC).

Microcontroller power consumption depends on chip size, supply voltage, clock frequency and operating mode. Embedded applications can optimize STM32 MCU power consumption by reducing the clock frequency when fast processing is not required and choosing the optimal operating mode and voltage range to run from. A description of STM32 power modes and voltage range is provided below.

## D.1 Power modes

STM32 MCUs support different power modes (refer to STM32 MCU datasheets for full details).

### D.1.1 STM32L1 series

STM32L1 microcontrollers feature up to 6 power modes, including 5 low-power modes:

- **Run** mode

  This mode offers the highest performance using HSE/HSI clock sources. The CPU runs up to 32 MHz and the voltage regulator is enabled.

- **Sleep** mode

  This mode uses HSE or HSI as system clock sources. The voltage regulator is enabled and the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs.

- **Low- power run** mode

  This mode uses the multispeed internal (MSI) RC oscillator set to the minimum clock frequency (131 kHz) and the internal regulator in low-power mode. The clock frequency and the number of enabled peripherals are limited.

- **Low-power sleep** mode

  This mode is achieved by entering Sleep mode. The internal voltage regulator is in low-power mode. The clock frequency and the number of enabled peripherals are limited. A typical example would be a timer running at 32 kHz.

  When the wakeup is triggered by an event or an interrupt, the system returns to the Run mode with the regulator ON.

- **Stop** mode

  This mode achieves the lowest power consumption while retaining RAM and register contents. Clocks are stopped. The real-time clock (RTC) an be backed up by using LSE/LSI at 32 kHz/37 kHz. The number of enabled peripherals is limited. The voltage regulator is in low-power mode.

  The device can be woken up from Stop mode by any of the EXTI lines.

- **Standby** mode

  This mode achieves the lowest power consumption. The internal voltage regulator is switched off so that the entire $V_{CORE}$ domain is powered off. Clocks are stopped and the real-time clock (RTC) can be preserved up by using LSE/LSI at 32 kHz/37 kHz.

RAM and register contents are lost except for the registers in the Standby circuitry. The number of enabled peripherals is even more limited than in Stop mode.

The device exits Standby mode upon reset, rising edge on one of the three WKUP pins, or if an RTC event occurs (if the RTC is ON).

*Note: When exiting Stop or Standby modes to enter the Run mode, STM32L1 MCUs go through a state where the MSI oscillator is used as clock source. This transition can have a significant impact on the global power consumption. For this reason, STM32CubeMX PCC introduces two transition steps:* ***WU_FROM_STOP*** *and* ***WU_FROM_STANDBY****. During these steps, the clock is automatically configured to MSI.*

### D.1.2 STM32F4 series

STM32F4 microcontrollers feature a total of 5 power modes, including 4 low-power modes:

- **Run** mode

  This is the default mode at power-on or after a system reset. It offers the highest performance using HSE/HSI clock sources. The CPU can run at the maximum frequency depending on the selected power scale.

- **Sleep** mode

  Only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/even occurs. The clock source is the clock that was set before entering Sleep mode.

- **Stop** mode

  This mode achieves a very low power consumption using the RC oscillator as clock source. All clocks in the 1.2 V domain are stopped as well as CPU and peripherals. PLL, HSI RC and HSE crystal oscillators are disabled. The content of registers and internal SRAM are kept.

  The voltage regulator can be put either in normal Main regulator mode (MR) or in Low-power regulator mode (LPR). Selecting the regulator in low-power regulator mode increases the wakeup time.

  The Flash memory can be put either in Stop mode to achieve a fast wakeup time or in Deep power-down to obtain a lower consumption with a slow wakeup time.

  The Stop mode features two sub-modes:

  – **Stop in Normal mode (default mode)**

    In this mode, the 1.2 V domain is preserved in nominal leakage mode and the minimum V12 voltage is 1.08 V.

  – **Stop in Under-drive mode**

    In this mode, the 1.2 V domain is preserved in reduced leakage mode and V12 voltage is less than 1.08 V. The regulator (in Main or Low-power mode) is in under-drive or low-voltage mode. The Flash memory must be in Deep-power-down mode. The wakeup time is about 100 µs higher than in normal mode.

- **Standby** mode

  This mode achieves very low power consumption with the RC oscillator as a clock source. The internal voltage regulator is switched off so that the entire 1.2 V domain is powered off: CPU and peripherals are stopped. The PLL, the HSI RC and the HSE crystal oscillators are disabled. SRAM and register contents are lost except for registers in the backup domain and the 4-byte backup SRAM when selected. Only RTC and LSE oscillator blocks are powered. The device exits Standby mode when an external reset (NRST pin), an IWDG reset, a rising edge on the WKUP pin, or an RTC alarm/ wakeup/ tamper/time stamp event occurs.

- **$V_{BAT}$** operation

  It allows to significantly reduced power consumption compared to the Standby mode. This mode is available when the $V_{BAT}$ pin powering the Backup domain is connected to an optional standby voltage supplied by a battery or by another source. The $V_{BAT}$ domain is preserved (RTC registers, RTC backup register and backup SRAM) and RTC and LSE oscillator blocks powered. The main difference compared to the Standby mode is external interrupts and RTC alarm/events do not exit the device from $V_{BAT}$ operation. Increasing $V_{DD}$ to reach the minimum threshold does.

### D.1.3 STM32L0 series

STM32L0 microcontrollers feature up to 8 power modes, including 7 low-power modes to achieve the best compromise between low-power consumption, short startup time and available wakeup sources:

- **Run mode**

  This mode offers the highest performance using HSE/HSI clock sources. The CPU can run up to 32 MHz and the voltage regulator is enabled.

- **Sleep mode**

  This mode uses HSE or HSI as system clock sources. The voltage regulator is enabled and only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs.

- **Low-power run mode**

  This mode uses the internal regulator in low-power mode and the multispeed internal (MSI) RC oscillator set to the minimum clock frequency (131 kHz). In Low-power run mode, the clock frequency and the number of enabled peripherals are both limited.

- **Low-power sleep mode**

  This mode is achieved by entering Sleep mode with the internal voltage regulator in low-power mode. Both the clock frequency and the number of enabled peripherals are limited. Event or interrupt can revert the system to Run mode with regulator on.

- **Stop mode with RTC**

  The Stop mode achieves the lowest power consumption with, while retaining the RAM, register contents and real time clock. The voltage regulator is in low-power mode. LSE or LSI is still running. All clocks in the $V_{CORE}$ domain are stopped, the PLL, MSI RC, HSE crystal and HSI RC oscillators are disabled.

  Some peripherals featuring wakeup capability can enable the HSI RC during Stop mode to detect their wakeup condition. The device can be woken up from Stop mode by any of the EXTI line, in 3.5 µs, and the processor can serve the interrupt or resume the code.

- **Stop mode without RTC**

  This mode is identical to “Stop mode with RTC “, except for the RTC clock which is stopped here.

- **Standby mode with RTC**

  The Standby mode achieves the lowest power consumption with the real time clock running. The internal voltage regulator is switched off so that the entire $V_{CORE}$ domain is powered off. The PLL, MSI RC, HSE crystal and HSI RC oscillators are also switched off. The LSE or LSI is still running.

After entering Standby mode, the RAM and register contents are lost except for registers in the Standby circuitry (wakeup logic, IWDG, RTC, LSI, LSE Crystal 32 KHz oscillator, RCC_CSR register).

The device exits Standby mode in 60 µs when an external reset (NRST pin), an IWDG reset, a rising edge on one of the three WKUP pins, RTC alarm (Alarm A or Alarm B), RTC tamper event, RTC timestamp event or RTC Wakeup event occurs.

- **Standby mode without RTC**

  This mode is identical to Standby mode with RTC, except that the RTC, LSE and LSI clocks are stopped.

  The device exits Standby mode in 60 µs when an external reset (NRST pin) or a rising edge on one of the three WKUP pin occurs.

*Note: The RTC, the IWDG, and the corresponding clock sources are not stopped automatically by entering Stop or Standby mode. The LCD is not stopped automatically by entering Stop mode.*

## D.2 Power consumption ranges

STM32 MCUs power consumption can be further optimized thanks to the dynamic voltage scaling feature: the main internal regulator output voltage V12 that supplies the logic (CPU, digital peripherals, SRAM and Flash memory) can be adjusted by software by selecting a power range (STM32L1 and STM32L0) or power scale (STM32 F4).

Power consumption range definitions are provided below (refer to STM32 MCU datasheets for full details).

### D.2.1 STM32L1 series feature 3 $V_{CORE}$ ranges

- High Performance **Range 1** ($V_{DD}$ range limited to 2.0-3.6 V), with the CPU running at up to 32 MHz

  The voltage regulator outputs a 1.8 V voltage (typical) as long as the $V_{DD}$ input voltage is above 2.0 V. Flash program and erase operations can be performed.
- Medium Performance **Range 2** (full $V_{DD}$ range), with a maximum CPU frequency of 16 MHz

  At 1.5 V, the Flash memory is still functional but with medium read access time. Flash program and erase operations are still possible.
- Low Performance **Range 3** (full $V_{DD}$ range), with a maximum CPU frequency limited to 4 MHz (generated only with the multispeed internal RC oscillator clock source)

  At 1.2 V, the Flash memory is still functional but with slow read access time. Flash Program and erase operations are no longer available.

### D.2.2 STM32F4 series feature several $V_{CORE}$ scales

The scale can be modified only when the PLL is OFF and when HSI or HSE is selected as system clock source.

- **Scale 1** (V12 voltage range limited to 1.26-1.40 V), default mode at reset
  HCLK frequency range = 144 MHz to 168 MHz (180 MHz with over-drive).
  This is the default mode at reset.
- **Scale 2** (V12 voltage range limited to 1.20 to 1.32 V)
  HCLK frequency range is up to 144 MHz (168 MHz with over-drive)
- **Scale 3** (V12 voltage range limited to 1.08 to 1.20 V), default mode when exiting Stop mode
  HCLK frequency ≤120 MHz.

The voltage scaling is adjusted to $f_{HCLK}$ frequency as follows:

- **STM32F429x/39x MCUs**:
  - **Scale 1:** up to 168 MHz (up to 180 MHz with over-drive)
  - **Scale 2:** from 120 to 144 MHz (up to 168 MHz with over-drive)
  - **Scale 3:** up to 120 MHz.
- **STM32F401x MCUs**:
  No Scale 1
  - **Scale 2:** from 60 to 84 MHz
  - **Scale 3:** up to 60 MHz.
- **STM32F40x/41x MCUs**:
  - **Scale 1:** up to 168 MHz
  - **Scale 2:** up to 144 MHz

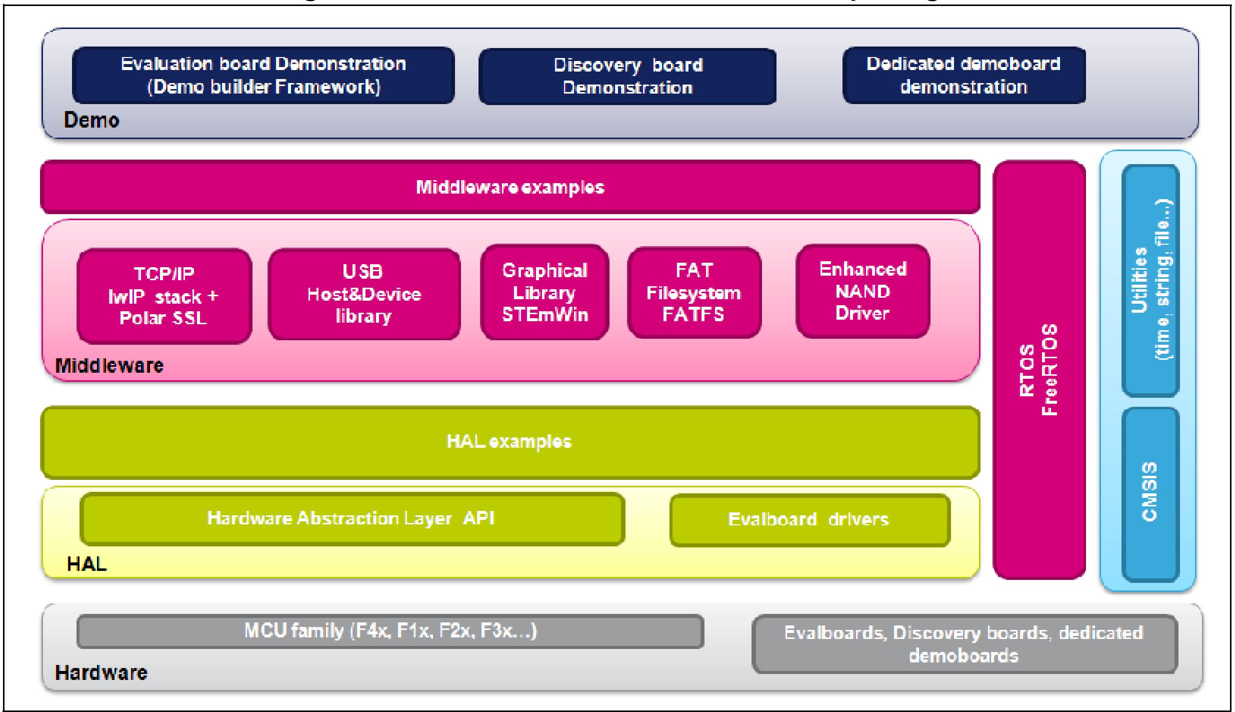
### D.2.3 STM32L0 series feature 3 $V_{CORE}$ ranges

- Range 1 ($V_{DD}$ range limited to 1.71 to 3.6 V), with CPU running at a frequency up to 32 MHz
- Range 2 (full $V_{DD}$ range), with a maximum CPU frequency of 16 MHz
- Range 3 (full $V_{DD}$ range), with a maximum CPU frequency limited to 4.2 MHz.

# Appendix E STM32Cube embedded software packages

Along with STM32CubeMX C code generator, embedded software packages are part of STM32Cube initiative (refer to *DB2164 databrief*): these packages include a low level hardware abstraction layer (HAL) that covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards (see *Figure 155*). This set of components is highly portable across the STM32 series. The packages are fully compatible with STM32CubeMX generated C code.

**Figure 155. STM32Cube Embedded Software package**




*Note:* *STM32CubeF0, STM32CubeF1, STM32CubeF2, STM32CubeF3, STM32CubeF4, STM32CubeL0 and STM32CubeL1 embedded software packages are available on st.com. They are based on STM32Cube release v1.1 (other series will be introduced progressively) and include the embedded software libraries used by STM32CubeMX for initialization C code generation.*

*The user should use STM32CubeMX to generate the initialization C code and the examples provided in the package to get started with STM32 application development.*

# 10 Revision history

**Table 15. Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 17-Feb-2014 | 1 | Initial release. |
| 04-Apr-2014 | 2 | Added support for STM32CubeF2 and STM32F2 series in cover page, *Section 2.2: Key features*, *Section 4.11.1: IP and Middleware Configuration window*, and *Appendix E: STM32Cube embedded software packages*.<br>Updated *Section 6.1: Creating a new STM32CubeMX Project*, *Section 6.2: Configuring the MCU pinout*, *Section 6.6: Configuring the MCU initialization parameters*.<br>Section "Generating GPIO initialization C code move to *Section 8: Tutorial 3- Generating GPIO initialization C code (STM32F1 series only)* and content updated.<br>Added *Section 9.4: Why do I get the error "Java 7 update 45' when installing 'Java 7 update 45' or a more recent version of the JRE?*. |
| 24-Apr-2014 | 3 | Added support for STM32CubeL0 and STM32L0 series in cover page, *Section 2.2: Key features*, *Section 2.3: Rules and limitations* and *Section 4.11.1: IP and Middleware Configuration window*<br>Added board selection in *Table 2: File menu functions*, *Section 4.4.3: Pinout menu* and *Section 4.2: New project window*. Updated *Table 4: Pinout menu*.<br>Updated *Figure 53: Power consumption calculator default view* and added battery selection in *Section 4.13.1: Building a power consumption sequence*.<br>Updated note in *Section 4.13: Power Consumption Calculator (PCC) view*<br>Updated *Section 6.1: Creating a new STM32CubeMX Project*.<br>Added *Section 9.5: Why does the RTC multiplexer remain inactive on the Clock tree view?*, *Section 9.6: How can I select LSE and HSE as clock source and change the frequency?*, and *Section 9.7: Why STM32CubeMX does not allow me to configure PC13, PC14, PC15 and PI8 as outputs when one of them is already configured as an output?*. |

**Table 15. Document revision history (continued)**

| Date | Revision | Changes |
|---|---|---|
| 19-jun-2014 | 4 | Added support for STM32CubeF0, STM32CubeF3, STM32F0 and STM32F3 series in cover page, *Section 2.2: Key features*, *Section 2.3: Rules and limitations*,<br>Added board selection capability and pin locking capability in *Section 2.2: Key features*, *Table 1: Welcome page shortcuts*, *Section 4.2: New project window*, *Section 4.4: Toolbar and menus*, *Section 4.6: Set unused / Reset used GPIOs windows*, *Section 4.7: Project Settings Window*, and *Section 4.10: Pinout view*. Added *Section 4.10.5: Pinning and labeling signals on pins*.<br>Updated *Section 4.11: Configuration view* and *Section 4.12: Clock tree configuration view* and *Section 4.13: Power Consumption Calculator (PCC) view*.<br>Updated *Figure 17: STM32CubeMX Main window upon MCU selection*, *Figure 27: Project Settings window*, *Figure 30: About window*, *Figure 31: STM32CubeMX Pinout view*, *Figure 32: Chip view*, *Figure 53: Power consumption calculator default view*, *Figure 54: Battery selection*, *Figure 55: Building a power consumption sequence*, *Figure 59: Power consumption sequence: new step default view (STM32F4 example)*, *Figure 60: Power Consumption Calculator view after sequence building*, *Figure 62: Sequence table management functions*, *Figure 63: STM32L0 PCC step edited in Edit Step window (STM32L0 example)*, *Figure 64: Power consumption sequence: new step configured (STM32F4 example)*, *Figure 65: ADC selected in Pinout view*, *Figure 66: PCC Step configuration window: ADC enabled using import pinout*, *Figure 68: Description of the result section*, *Figure 69: Peripheral power consumption tooltip*, *Figure 131: Power Consumption Calculation example*, *Figure 132: Sequence table* and *Figure 133: Power Consumption Calculation results*.<br>Updated *Figure 39: STM32CubeMX Configuration view* and *Figure 39: STM32CubeMX Configuration view - STM32F1 series* titles.<br>Added STM32L1 in *Section 4.13: Power Consumption Calculator (PCC) view*.<br>Removed Figure *Add a new step using the PCC panel* from *Section 8.1.1: Adding a step*. Removed Figure *Add a new step to the sequence* from *Section 8.2: Configuring a step in the power sequence*.<br>Updated *Section 8.3: Reviewing results*.<br>Updated appendix *B.3.4: FATFS* and *Appendix D: STM32 microcontrollers power consumption parameters*. Added Appendix *D.1.3: STM32L0 series* and *D.2.3: STM32L0 series feature 3 VCORE ranges*. |

**Table 15. Document revision history (continued)**

| Date | Revision | Changes |
|---|---|---|
| 19-Sep-2014 | 5 | Added support for STM32CubeL1 series in cover page, *Section 2.2: Key features*, *Section 2.3: Rules and limitations*,<br>Updated *Section 3.2.2: Uninstalling STM32CubeMX standalone version*.<br>Added off-line updates in *Section 3.5: Getting STM32Cube updates*, modified *Figure 13: New libraries Manager window*, and *Section 3.5.2: Downloading new libraries*.<br>Updated *Section 4: STM32CubeMX User Interface* introduction, *Table 1: Welcome page shortcuts* and *Section 4.2: New project window*.<br>Added *Figure 16: New Project window - board selector*.<br>Updated *Figure 29: Project Settings Code Generator*.<br>Modified step 3 in *Section 4.7: Project Settings Window*.<br>Updated *Figure 39: STM32CubeMX Configuration view - STM32F1 series*.<br>Added STM32L1 in *Section 4.11.1: IP and Middleware Configuration window*.<br>Updated *Figure 42: GPIO Configuration window - GPIO selection*; *Section 4.11.2: GPIO Configuration window* and *Figure 48: DMA MemToMem configuration*.<br>Updated introduction of *Section 4.12: Clock tree configuration view*. Updated *Section 4.12.1: Clock tree configuration functions* and *Section 4.12.2: Recommendations*, *Section 4.13: Power Consumption Calculator (PCC) view*, *Figure 59: Power consumption sequence: new step default view (STM32F4 example)*, *Figure 60: Power Consumption Calculator view after sequence building*, *Figure 64: Power consumption sequence: new step configured (STM32F4 example)*, and *Figure 66: PCC Step configuration window: ADC enabled using import pinout*. Added *Figure 67: Power Consumption: Peripherals Consumption Chart* and updated *Figure 69: Peripheral power consumption tooltip*. Updated *Section 4.13.3: Power sequence step parameters glossary*.<br>Updated *Section 5: STM32CubeMX C Code generation overview*.<br>Updated *Section 6.1: Creating a new STM32CubeMX Project* and *Section 6.2: Configuring the MCU pinout*.<br>Added *Section 7: Tutorial 2 - Example of FATFS on an SD card using STM32F429I-EVAL evaluation board* and updated *Section 8: Tutorial 3- Generating GPIO initialization C code (STM32F1 series only)*.<br>Updated *Section 8.2: Configuring a step in the power sequence*. |

**Table 15. Document revision history (continued)**

| Date | Revision | Changes |
|---|---|---|
| 19-Jan-2015 | 6 | Complete project generation, power consumption calculation and clock tree configuration now available on all STM32 series.<br>Updated *Section 2.2: Key features* and *Section 2.3: Rules and limitations*.<br>Updated Eclipse IDEs in *Section 3.1.3: Software requirements*.<br>Updated *Figure 9: Updater Settings tab*, *Figure 13: New libraries Manager window* and *Figure 16: New Project window - board selector*, Updated *Section 4.7: Project Settings Window* and *Section 4.8: Update Manager Windows*.<br>Updated *Figure 30: About window*.<br>Removed Figure *STM32CubeMX Configuration view - STM32F1 series.*<br>Updated *Table 8: STM32CubeMX Chip view - Icons and color scheme*.<br>Updated *Section 4.11.1: IP and Middleware Configuration window*.<br>Updated *Figure 46: Adding a new DMA request* and *Figure 48: DMA MemToMem configuration*.<br>Updated *Section 4.12.1: Clock tree configuration functions*.<br>Updated *Figure 54: Battery selection*, *Figure 55: Building a power consumption sequence*, *Figure 63: STM32L0 PCC step edited in Edit Step window (STM32L0 example)*.<br>Added *Section 5.2: Custom code generation*.<br>Updated *Figure 84: Clock tree view* and *Figure 89: Configuration view*.<br>Updated peripheral configuration sequence and *Figure 91: Timer 3 configuration window* in *Section 6.6.2: Configuring the peripherals* .<br>Removed Tutorial 3: Generating GPIO initialization C code (STM32F1 series only).<br>Updated *Figure 94: GPIO mode configuration*.<br>Updated *Figure 131: Power Consumption Calculation example* and *Figure 132: Sequence table*.<br>Updated Appendix *A.1: Block consistency*, *A.2: Block inter-dependency* and *A.3: One block = one peripheral mode*.<br>Appendix *A.4: Block remapping (STM32F10x only)*: updated *Section : Example* .<br>Appendix *A.6: Block shifting (only for STM32F10x and when "Keep Current Signals placement" is unchecked)*: updated *Section : Example*<br>Updated Appendix *A.8: Mapping a function individually* .<br>Updated Appendix *B.3.1: Overview*.<br>Updated Appendix *D.1.3: STM32L0 series*. |

Table 15. Document revision history (continued)

| Date | Revision | Changes |
|---|---|---|
| 19-Mar-2015 | 7 | *Section 2.2: Key features*: removed *Pinout initialization C code generation for STM32F1 series* from; updated *Complete project generation*.<br>Updated *Figure 13: New libraries Manager window*, *Figure 16: New Project window - board selector*.<br>Updated IDE list in *Section 4.7: Project Settings Window* and modified *Figure 27: Project Settings window*.<br>Updated *Section 4.12.1: Clock tree configuration functions*. Updated *Figure 50: STM32F429xx Clock Tree configuration view*.<br>*Section 4.13: Power Consumption Calculator (PCC) view*: added transition checker option. Updated *Figure 53: Power consumption calculator default view*, *Figure 54: Battery selection* and *Figure 55: Building a power consumption sequence*. Added *Figure 56: Enabling the transition checker option on an already configured sequence - all transitions valid*, *Figure 57: Enabling the transition checker option on an already configured sequence - at least one transition invalid* and *Figure 58: Transition checker option -show log*. Updated *Figure 60: Power Consumption Calculator view after sequence building*. Updated *Section : Managing sequence steps*, *Section : Managing the whole sequence*. Updated *Figure 63: STM32L0 PCC step edited in Edit Step window (STM32L0 example)* and *Figure 68: Description of the result section*.<br>Updated *Figure 131: Power Consumption Calculation example*, *Figure 132: Sequence table*, *Figure 133: Power Consumption Calculation results* and *Figure 135: Power consumption results - IP consumption chart*.<br>Updated Appendix *B.3.1: Overview* and *B.3.5: FreeRTOS*. |