A Mini Reference Manual of STM32CubeMX

Jianhua Liu

v0.2, 3/22/2024

Contents

1 Introduction

STM32CubeMX, called CubeMX in the sequel for simplicity, is a configuration and automatic code generation tool for STM32 MCUs (including both MP cores and peripherals).

In addition to generating MCU initialization code based on the configuration of MCU peripherals, it can also generate middleware code based on the middleware selections, such as USB device descriptors. These initialization and middleware functions are called in the generated `main.c` function of the project so that the framework of the project is created.

There are guarded sections in the generated code where users can add their application-oriented code. Later, with the growing of the project, more hardware peripheral and middleware code can be added to the project without affecting the user code placed in these guarded sections.

CubeMX is a cross-platform tool, which can be used on Windows, MacOS, and Linux.

CubeMX is a complicated tool with a 400-page manual (UM1718), which can be downloaded from https://www.st.com/en/development-tools/stm32cubemx.html#documentation. Here, we provide a very short reference manual so that we can easily find the most useful information of how to use this tool effectively.

1.1 CubeMX vs STM32CubeIDE

Note that CubeMX is already integrated into STM32CubeIDE, referred to CubeIDE for simplicity. Yet, for reasons listed below, we plan to use the standalone version of CubeMX.

- Many times, we want to be able to generate code for multiple build tools, including CubeIDE and Keil. The integrated CubeMX cannot generate code for Keil; see page 3 of www.keil.com/appnotes/docs/apnt_323.asp. The reason is that STM32CubeIDE organizes the projects in its own workspace as it is Eclipse-based.
- The integrated CubeMX does not support a clean structure of the CubeIDE project files—all the files are generated at the root of the project. The standalone version of CubeMX can generate the CubeIDE project files within a folder, which reduces the clutter.

The class projects are generated using CubeMX, and you are required to follow the same approach for class projects. If you create your own projects without considering class submission, you can use the CubeMX tool within CubeIDE.

2 Installation of CubeMX and software packages

2.1 Installing CubeMX

CubeMX can be downloaded from https://www.st.com/en/development-tools/stm32cubemx.html. When you click to download the appropriate version, you will see a pop-up **License Agreement** page. You need to enter info in order to download the software (you may have to wait for a confirmation email).

In order to ensure that there are no issues with project generation later on in the course, please ensure that you are downloading and installing version 6.8.1. Higher version will be ok if you install the corresponding version of CubeIDE. (CubeMX v6.8.1 corresponds to CubeIDE 1.12.1.)

The installation process is fairly straight forward and intuitive.

- For windows, we can just install the tool by default at

  ```
  C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeMX
  ```

  following the hint and use all default choices.
- For macOS, . . . (to be added later)

2.2 Installing embedded software packages

To generate code for specific MCUs using CubeMX, we need to install their software packages, which also include a large number of application examples.

We use a number of different MPBs for the projects of the class. They belong to STM32F4 and STM32G4 families. As such, we need to install there software packages. This can be done in CubeMX by following the steps below:

- Click **[Help » Manage embedded software packages]**.
- Click the [**down arrow**] beside the [**STM32F4**] node and choose version **1.27.1**. This is required as we will build programs based on STM32F412ZG to run on Renode.
- Repeat the above to install version 1.5.1 for *STM32G4*. This is required if you use G431n32.

If you use other MCUs from other families, you need to install the corresponding software package as well. For example, if you use L432n32, you can install version 1.17.2 for *STM32L4*.

2.3 Adding MCUs to favorites (optional)

In cases where you need to make new projects very often from scratch, it may be useful to have the devices you use often saved in the *favorites* category in CubeMX so that they can be found faster.

To do this, go to **[File » New project]**, enter the following device name, such as `stm32f412zg` in the *Part Number* search bar, and click the *star* on the far left of the item. Details of searching a part number can be found at the Selecting a device section below.

2.4 Setting up the updater

To set up the updater, use the following in CubeMX.

- Click **[Help » Updater Settings . . .]** to bring up the **Updater Settings** dialog.
- In the [**Updating Settings**] tab of the dialog, we can change the settings, including the folder of repository. Here, we just leave them as is.

If we want to manually update CubeMX, we can use **[Help » Check for updates]**.

3 Selecting a device for a new project

Every new project starts at selecting a device in CubeMX.

3.1 Selecting a device

Now, we assume CubeMX is up and running and the **Home** page is on focus, as shown in Fig. 1. If not, just click **[Home]** at the top left corner of the window and it will be shown (Need to update the figure to a newer version later).

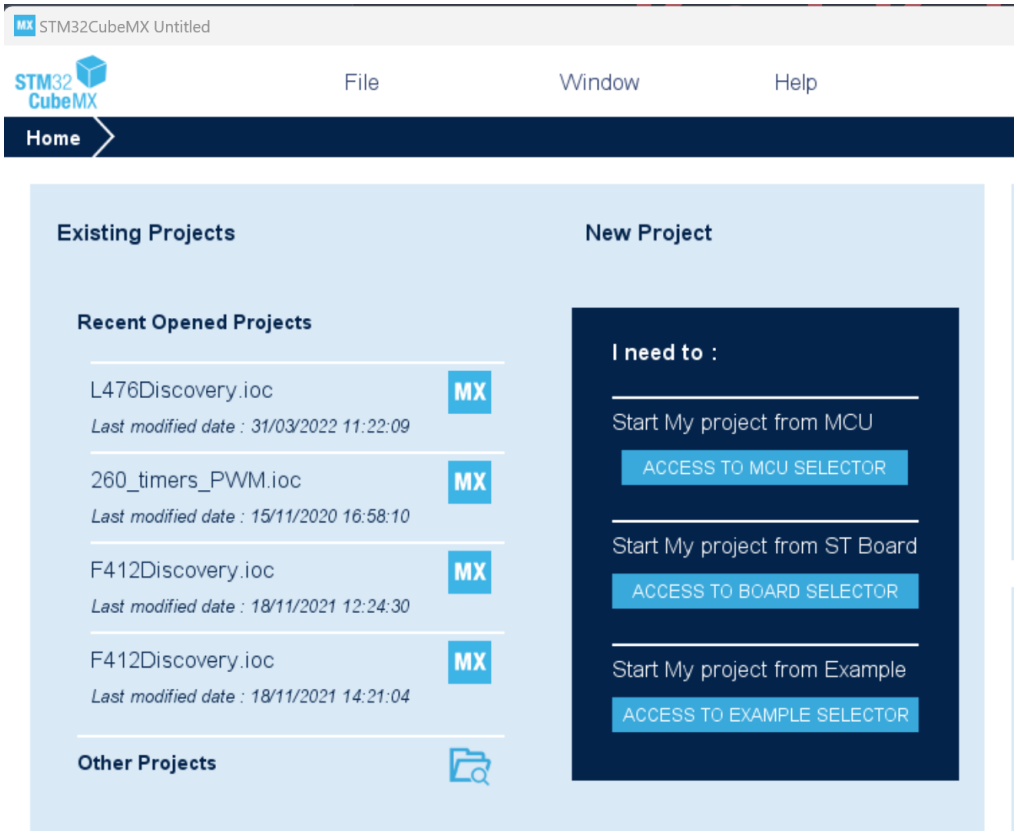

Figure 1: The "Home" window of STM32CubeMX.

At the *Home* page, the following two choices in the center column are most useful for us:

- **ACCESS TO MCU SELECTOR**
- **ACCESS TO BOARD SELECTOR**

If we want to build a project using all the functionalities of a certain ST MPB, use the second choice. Otherwise, use the first one.

Whichever choice we make, we will see a large selection. At the time of writing, we have:

- 3500+ MCU selections
- 170+ MPB selections

While we may be able to browse the ST MPBs, it is very difficult to browse the selection of MCUs. What we usually do is to search by **Part Number** or features.

- If we know part of the Part Number, such as `F412`, we can type it in the **Part Number** search box or click the dropdown in the search box to select.
- If we don't know the part number but we know what we want, such as the size of Flash and RAM or the specific peripheral, we can narrow the selection using the choice selectors.

3.2 Default pins of a MCU

Assume we have selected the MCU or MPB, and the **Pinout & Configuration** tab is shown at the center of CubeMX.

If an MCU is chosen, the MCU is shown without pins preassigned except the following:

- VDD, which stands for the voltage of the *drain* of an FET (field effect transistor). This is the normal power supply of the MCU, which usually has a value of 3.3 V, often denoted as 3V3.
- VSS, which stands for the voltage of the *source* of an FET. This is the normal ground of the MCU.
- NRST (nRST), which stands for *negated* Reset. This is the reset pin of the MCU, which is effective when it is low; that is why we call it negated.
- BOOT0, which stands for the boot pin 0. By using this pin, we can choose to where to boot the MCU when the NRST is low. This is very useful as we can update the firmware of the MCU without using an in-circuit debugger if we choose to boot from the system memory.

Now, we can set up peripheral and pins as needed.

3.3 How about the preassigned pins when an MPB is selected

If an MPB is chosen, the MCU is shown with many pins preassigned. We can still add peripherals and IO pins to those not used as we do for a normal MCU. Yet, we should be very careful when reassigning those already used pins, as this may create conflicts with the existing onboard hardware or even damage some components.

4 Saving a project

As we discussed in Supporting multiple MPBs in a single project, we will support multiple MPBs in a single project. As such, we need to have a folder structure shown in Organization of projects. Now, we use the `cc3d_uart_printf_direct` project as an example to illustrate this.

Assume we have already created the `cc3d_uart_printf_redirect` folder, which is the root folder of the project used to host all the files for this project. Now, assume we use F412dsc, and we want to save the files related to this MPB to the `cc3d_uart_printf_f412dsc` folder under the root folder. (The reason we repeat part of the root folder's name is for the display purposes in CubeIDE.)

Now, we can save the project with the following steps:

- Click [File » Save Project As . . .]. This brings up the **Save Project As** dialog.
- Browse to the `cc3d_uart_printf_f412dsc` folder to save the project. If it not created yet, create it in this dialog by right-clicking to bring out the context menu and following the hint.

Note that the CubeMX project files have an extension of .ioc, and the project file should have the same name as the folder where it is saved.

In case you want to know the meaning of **ioc**, we can think of it means *input and output configuration*, at least we can understand it this way until we find the true meaning.

5 Setting up SWD and SWV

All the MPBs we use have an onboard debugger, called ST-Link, which has the following functionalities:

- Debugging using SWD (serial wire debug), an improvement of the traditional JTAG protocols. This uses two wires on the MCU: SW_CLK (clock) and SW_DIO (two way, input and output, data).
- Tracing with SWO (serial wire output). This uses an additional wire in addition to SWD to provide single wire tracing to provide more information about the running of the MCU.

To run a program on the MCU only, we don't need to set up the SWD probe. Yet, without setting up SWD, we cannot use the debugger function to step through the code. So, we normally set up SWD for all projects. If we need to log the running info of the program without slowing down it too much, we need to use SWV.

Assume we are in the **Pinout & Configuration** tab. We just need to follow the steps below for a quick setup:

- Go to the **Categories** tab in the left pane of the **Pinout & Configuration** tab.
- Open up the **System Core** node and click on **[SYS]**. Now, we should see a **SYS Mode and Configuration** pane to the right of the pane where the **Categories** tab is displayed. If not, we need to click on the *right* arrow at the top of the pane divider.
- In the dialog, set the **Debug** dropdown to:
  - *Serial Wire* if we only want use the SWD function.
  - *Trace Asynchronous Sw* if we want to use SWV for tracing.

Note that for SWV, we need to perform additional setup in CubeIDE, as shown in Using SWV in CubeIDE.

6 Setting up USART2 for VCP

USART2, used in the UART mode, is connected to the onboard ST-Link V2.1 probe, which can be used to bridge this UART signal to PC using VCP.

Here we discuss how to use CubeMX to set up USART2 so that we can use a Serial Com terminal tool, such as HTerm, on the PC to establish communications between the PC and the MCU via VCP.

Note that the setup here is for simple basic UART operations. Later, we will discuss how to set up the UART to use interrupt and DMA to improve efficiency.

Before we do any setup, we need to specify pins used for connecting USART2 to ST-Link on the MPBs. as there may be multiple different pins that can be connected to USART2 on the MCU. We need to use the pins listed below for the MPBs we use:

Pin label (Function)	F412dsc	G431n32	L432n32	L476dsc
VCP_TX	PA2	PA2	PA2	PD5
VCP_RX	PA3	PA3	PA15	PD6

We assume that we have chosen the MCU and saved the project file, as illustrated by the instructions of the above sections. We also assume that CubeMX is the **Pinout & Configuration** tab. Now, we can move forward to set up the pins and the parameters of USART2.

Note that the definitions of the above pins are found in the schematics of the boards we use. For example:

- For F412dsc, the connections are given on pages 44/48 in user manual, UM2032 (version 2).
- For L476dsc the connections are given on pages 26 and 27 in version 4 of UM1879.

There are a number of setups we need to go over; there are different paths to take. Here, we illustrate a relatively easy one. If we are lucky enough, there is only one step for the setup except pin labeling.

Below, we illustrate the steps using F412dsc. Again, we assume we are in the **Pinout & Configuration** tab.

6.1 Enabling USART2

- Go to the **Categories** tab in the left pane of the **Pinout & Configuration** tab.
- Open up the **Connectivity** node and click on **[USART2]**. Now, we should see a **USART2 Mode and Configuration** pane to the right of the pane where the **Categories** tab is displayed. If not, we need to click on the *right* arrow at the top of the pane divider.
- In the **Mode** field of this pane, do the following:
  - In the **Mode** dropdown, select the *Asynchronous* mode.
  - In the **Hardware Flow Control (RS232)** dropdown, select *Disable*.
  - Do not check **Hardware Flow Control (RS485)** if there is such a box.

Now, USART2 is enabled, and two pins are chosen for us for data transmitting and receiving.

6.2 Choosing and labeling USART2 pins

We need to check if the suggested pins are the same as what we want. If yes, no need to change. If not, we can ask CubeMX to suggest alternative pins as described at Seeing the alternative pins. Choose the alternatives that are listed above.

Now, we need to change the labels of the TX and RX pins to `VCP_TX` and `VCP_RX`, respectively. While this is optional for a personal project, it is a must for everyone in the class so that we can be consistent.

6.3 Configuring GPIOs

The default configurations are what we need. To see the GPIO configurations, make sure we use the **Configuration** field of the **USART2 Mode and Configuration** pane. Click the **GPIO Settings** tab to make sure both of the above two pins have the following configurations:

- **GPIO Mode**: *Alternate Function Push Pull*.
- **GPIO Pull-up/Pull-down**: *No pull-up and no pull-down*.
- **Maximum output speed**: *Very High*.

6.4 Configuring communication parameters

Again, the default configurations are what we need. To see the communication parameter configurations, click the **Parameter Settings** tab in the above **Configuration** field. Make sure we have the following:

- In the **Basic Parameters** group:
  - **Baud Rate**: *115200 Bits/s*
  - **Word Length**: *8 Bits (including Parity)*
  - **Parity**: *None*
  - **Stop Bits**: *1*
- In the **Advanced Parameters** group:
  - **Data Direction**: *Receive and Transmit*.
  - **Over Sampling**: *16 Samples*.
  - **Single Sample** (if this appears): *Disable*.

Note that for a simple project like this, we don't need to set up the clock tree and debug prob. The above is all we need to set up for the MCU and middleware. We can move on to generate code for different IDEs.

6.5 Configuring interrupt for USART2

If we want to use interrupt mode of USART2, we need to do this additional setting.

In the **Configuration** field of the **USART2 Mode and Configuration** pane, click the **NVIC Settings** tab and check the **Enabled** checkbox for **USART2 global interrupt / USART2 wakeup . . . .**

By default, the preemption priority and sub-priority are both set up to 0. If we want to change these values, we need to do it as shown in the Preemption priority and sub-priority setup section below.

7 Generating code from CubeMX

Assume we have both MCU and middleware set up. It is now time to generate the code for the intended toolchains: CubeIDE or Keil.

Now, we assume we have `cc3d_uart_printf_f412dsc.ioc` opened in CubeMX and have the following folder structure with the saved `.ioc` file inside:

```
cc3d_uart_printf_redirect
    cc3d_uart_printf_f412dsc
        cc3d_uart_printf_f412dsc.ioc
```

7.1 Configuring a project for code generation

Now, we can configure the code generation by clicking the **[Project Manager]** tab at the top area of CubeMX to switch to this tab. Then do the following:

- Select the **Project** tab (left side of the screen). There are 4 groups.
  - In the 1st group, the **Project Settings** group:

        * Leave the **Project Name** and **Project Location** fields unchanged.
        * Make sure to use *Basic* for the **Application Structure** field. (If there is no way to use *Basic*, just use **Save As** to save the .ioc file again to itself. The *Basic* option should be available.) Also make sure the **Do not generate the main()** box is unchecked.
        * Leave the **Toolchain Folder Location** field unchanged.
        * In the **Toolchain / IDE** dropdown, select the appropriate toolchain. Note that we need to uncheck **Generate Under Root** so that the toolchain related files will be in their own directories:
            · If you prefer to use CubeIDE, choose **CubeIDE**.
            · If you prefer to use Keil MDK, choose **MDK-ARM**. We can choose the **Min Version** as *V5.32*.
    - In the 2nd group, the **Linker Settings** group:
        * No need to change the **Minimum Heap Size** for a small project. For a larger one, we may need to increase it to `0x400` or `0x600`.
        * No need to change the **Minimum Stack Size** for a small project. For a larger one, we may need to increase it to `0x600`.
    - In the 3rd group, the **Thread-safe Settings** group:
        * Do not check the **Enable multi-threaded support** if we do not use RTOS.
    - In the 4th group, the **MCU and Firmware Package** group:
        * Leave the **MCU Reference** field unchanged.
        * Uncheck the **Use latest available version** checkbox. Select the version we have installed at Installing embedded software packages.
        * Check the **Use Default Firmware Location** checkbox.

- Select the **Code Generator** tab. Again, there are 4 groups:
    - In the 1st group, the **STM32Cube MCU packages and embedded software packs** group. Choose **Copy only the necessary library files**. (Note that we could choose **Add necessary library files as reference in the toolchain project configuration file** if we do not need to share the project often. With this choice, we don't need to copy the library source files to our project to save storage spaces.)
    - In the 2nd group, the **Generated files** group, check the following options:
        * **Keep user code when re-generating**.
        * **Delete previously generated files when not re-generated**.
    - In the 3rd group, the **HAL Settings** group, do the following option:
        * Check **Set all free pins as analog**.
        * Uncheck **Enable Full Assert**.
    - No need to change **Template Settings** in the 4th group.

- Select the **Advanced Settings** tab. There are two groups:
    - No need to change anything in the **Driver Selector** group. If we want to use low-level drivers instead of the HAL ones, we can make changes here.

  - In the **Generated Function Calls** group, make sure all the functions are visible. (This is needed to browse the generated code.)

7.2 Generating code

After all the above setups, we can finally generate code for the selected toolchain. This is easily done by clicking the **[GENERATE CODE]** button at the top right corner of the window.

When the code generation is done, a **Code Generation** dialog will pop up. Now, we have a number of choices:

- If we want to see the generated files, we can click **[Open Folder]**.
- If we want to open the project in the toolchain, we can click **[Open Project]**.
- Or, we can just click **[Close]** to keep working in CubeMX. For example, we can generate code for another toolchain.

After code is generated, we need to go to the toolchain to add user code and build and test the program. We need to continue at:

- For CubeIDE: Importing a project generated by CubeMX.
- For Keil MDK: TBD

8 Setting up GPIO pins for LEDs and push buttons

Here we illustrate how to use CubeMX to set up GPIO output pins used for driving LEDs and GPIO input pins used for reading buttons.

8.1 A summary of setup steps for GPIO output pins

Here are the steps for setting up GPIO output pins.

- Set each GPIO pin individually. This is done in the **Pinout** view.
  - Find the pin by searching.
  - Set the pin to `GPIO_Output` mode by clicking on it.
  - Label the pin by right-clicking on it and choose **[Enter User Label]**.
  - Repeat the above for all the output pins.
- Set the properties of all the GPIO pins together. This is done in the **Categories** tab in the left pane of the **Pinout & Configuration** tab.
  - Open up the **System Core** node and click on **[GPIO]**. Now, we should see a **GPIO Mode and Configuration** pane to the right of the pane where the **Categories** tab is displayed.
  - Set the initial **GPIO output level**: *High* or *Low*.
    * If the anode of the LED is connected to the GPIO pin, we need to set the output level to *Low* to turn off the LED at the beginning.
    * If the cathode of the LED is connected to the GPIO pin, we need to set the output level to *High* to turn off the LED at the beginning.

  - Choose the **GPIO mode**: *Output Push Pull* or *Output Open Drain*.
    * If the anode of the LED is connected to the GPIO pin, we need to use *Output Push Pull*.
    * If the cathode of the LED is connected to the GPIO pin, we need to use *Output Open Drain*.
  - Select the **Maximum output speed**: *Low* or *High*. For most cases, *Low* is faster enough than what we need. So use this choice.

8.2 A summary of setup steps for GPIO input pins

Here are the steps for setting up GPIO input pins.

The steps for setting up individual GPIO input pins are the same as those of individual GPIO output pins except we need to choose `GPIO_Input`.

The steps for setting up the properties of GPIO input pin together are the same as those for GPIO output pins except the following.

- Choose the **GPIO mode**: *Input mode*w, which is the default.
- Select the **GPIO Pull-up/Pull-down**:
  - *Pull-up* if the other side of the switch is connected to ground.
  - *Pull-down* if the other side of the switch is connected to VDD.

Note that detailed discussion will be provided later for each setup based on the similarity and difference of the two boards.

8.3 Detailed GPIO pin setup steps

Here, we use F412dsc and L476dsc to illustrate the detailed setups. The pin definitions can be found at Digital input pins and GPIO output and PWM output pins. The G431n32 and L432n32 boards can be set up similarly. Their definitions can be found at A Short Reference Manual for Hardware Configurations of Nucleo-32 Game Controller V1.0.

- Setting up the GPIO **output pins** used to drive the user LEDs on F412dsc. We need use the **Pinout** view.
  - Left-click **PE2** (for LED1) and **PE3** (for LED2) and **set them to `GPIO_Output` mode**. (If you cannot find a pin, type the pin number in the **Find/Search** box at the bottom of the window.)
  - Right-click each of the above pins and select **Enter User Label** to enter `LED1` and `LED2` for **PE2** and **PE2**, respectively.

- Setting up the GPIO **output pins** used to drive the user LEDs on L476dsc.
  - Left-click **PD0** (for LED1) and **PB2** (for LED2) and **set them to `GPIO_Output` mode**.
  - Right-click each of the above pins and select **Enter User Label** to enter `LED1` and `LED2` for **PD0** and **PB2**, respectively.

- Set up the **input pins** used to read the multi-directional switch on F412dsc. We still work in the above **Pinout** view. (Note that only 2 of the 5 pins are set up here for illustration purposes.)
  - Left-click **PF15** (for Nav2) and **PF14** (for Nav3) and **set them to `GPIO_Input` mode**.

  - Right-click each of the above pins and select **Enter User Label** to enter `Nav1` and `Nav2` for **PF15** and **PF14**, respectively.

- Set up the **input pins** used to read the multi-directional switch on L476dsc:

  - Left-click **PA1** (for Nav1) and **PA2** (for Nav2) and **set them to `GPIO_Input` mode**.
  - Right-click each of the above pins and select **Enter User Label** to enter `Nav1` and `Nav2` for **PA1** and **PA2**, respectively.

- Open the **GPIO Mode and Configuration** pane. This is done by opening up the **System Core** node and click on **[GPIO]**. Now, we should see a **GPIO Mode and Configuration** pane to the right of the pane where the **Categories** tab is displayed.

- Configure the other **GPIO Modes** in the **GPIO** tab in the **GPIO Mode and Configuration** pane. Note that we have different LED circuits for the two MPBs, as shown in Figure 2. This has to be considered when doing the setup. Now, click on each line of the GPIO pin and do the following separately.

  - Common modes for output pins (**PE2** and **PE3** for F412dsc or **PD0** and **PB2** for L476dsc):
    * **GPIO Pull-Up/Pull-Down**: *No Pull-Up and No Pull-Down* (This is output, so pull-up/pull-down does not matter much. Just use this default choice.)
    * **Maximum Output Speed**: *Low* (No need to use higher speed.)
  - Different modes for output pins (**PE2** and **PE3** for F412dsc or **PD0** and **PB2** for L476dsc):
    * **GPIO output level**: (LED will be off at the start.)
      · *High* for F412dsc as a low output will be able to create voltage difference between the anode and cathode of the LEDs as the anodes are connected to VDD (3V3).
      · *Low* for L476dsc as the anodes of the LED are connected to the GPIO output pins and a low output will not be able to drive a current through the LEDs.
    * **GPIO Mode**: (This include the true GPIO mode and the output type.)
      · *Output Open Drain* for F412dsc. (Push Pull can also be used, but is rare for this type of circuit.)
      · *Output Push Pull* for L476dsc. (Cannot use Open Drain since it cannot provide current to drive the LED on.)
  - Input pins: (**PF14** and **PF15** for STM32F412 or **PA1** and **PA2** for STM32L476):
    * **GPIO Mode**: *Input mode* as this is indeed for input.
    * **GPIO Pull-Up/Pull-Down**: *Pull-Down* as the other side of each switch is connected to the VDD. So when the switch is closed, the reading should be high; when the switch is not closed, the reading should then be *low*.

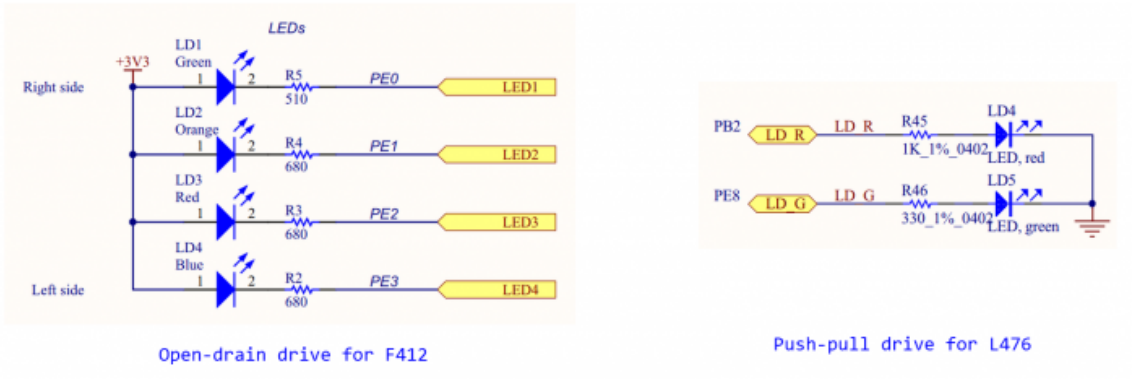

Figure 2: LED circuits for two HW kits.

9 Setting up interrupt priority with NVIC

There are two parts for the interrupt priority setup. The first is the priority group setting, and the second the detailed setup for preemption priority and sub-priority.

9.1 Setting up the priority group

Assume we are in the **Pinout & Configuration** tab. We just need to follow the steps below for a quick setup:

- Go to the **Categories** tab in the left pane of the **Pinout & Configuration** tab.
- Open up the **System Core** node and click on **[NVIC]**. Now, we should see a **NVIC Mode and Configuration** pane to the right of the pane where the **Categories** tab is displayed. If not, we need to click on the *right* arrow at the top of the pane divider.
- Click the **[NVIC]** tab at the top of the pane.
- At the top of this tab, we can find the **Priority Group** field. Click the dropdown and select needed number of bits for the **Priority Group** setting. Here, we use *2 bits for preemption* as a default set up for our projects. If a project requires other setting, we just do it here.

9.2 Setting up the preemption priority and sub-priority

Once the *Priority Group* setting is done, we can easily set up the preemption priority and sub-priority by clicking on the responding IRQn in the above pane (tab) and click to choose the appropriate numbers.

10 Setting up EXTI using CubeMX

There are two parts for the EXTI setup.

The first part is the same as the GPIO input setup; the only difference is that we use `EXTIx` instead of `GPIO_Input`. Here `x` is the pin number.

The second part is the setup of the NVIC. Let us use the setup of `Fnc6` as an example for G431n32 and L432n32.

- In the above **GPIO** tab, click the pin which is set up to as EXTI. This is PA0 on both G431n32 and L432n32. Select the **GPIO Mode** as *External Interrupt Mode with Rising edge trigger detection*. In this case, we only have a single interrupt trigger when the button is pressed. The other modes can be valid depending on design requirement.
- Now, go to the **NVIC** tab in the same **GPIO Mode and Configuration** pane. Check the **Enabled** checkbox for this **EXTI Line0 interrupt**.
- Next, open up the **System Core** node and click on **[NVIC]**. Now, we should see a **NVIC Mode and Configuration** pane. There are two tabs here.
  - Click the **NVIC** tab. We can see **EXTI Line0 interrupt** is *Enabled* and has 0 for both **Preemption Priority** and **Sub Priority**. These are good for many small projects: we need to have the interrupt enabled and for small projects, we don't need to have complex level priorities. For projects with more granular control of priority, we need to perform setup following the instructions of the Setting up interrupt priority section above.
  - Click the **Code Generation** tab. Have the following setups:
    * Uncheck the **Select for init sequence ordering** checkbox.
    * Check the **Generate Enable in Init** checkbox.
    * Check the **Generate IRQ handler** checkbox.
    * Check the **Call HAL handler** checkbox.

11 Setting up the clock tree

11.1 Basic clock tree setup

For many simple projects, there is no need to set up the clock tree. The default will use the following:

- Source of oscillator: HSI RC, 16 MHz.
- System Clock Mux: HSI
- AHB prescaler: 1

When we do need to set up the clock tree, we need to choose the source first. This is done in the **RCC Mode and Configuration** pane. To open this pane, we need to go to the **Categories** tab in the left pane of the **Pinout & Configuration** tab, open up the **System Core** node, and click on **[RCC]**. Now, we should see this pane, as shown in Figure 3. For many MCUs, there are three choices for the **High Speed Clock (HSE)** field:

- *Disable*, which means we will use the internal HSI. No GPIO pin is used for this case.
- *BYPASS Clock Source*, which means we will use a clock signal from the MCO (master clock output) of somewhere else. Here, we will need to use one IO pin to get the source in to the MCU.
- *Crystal/Ceramic Resonator*, which means an external crystal is needed. Two pins are needed for this case.

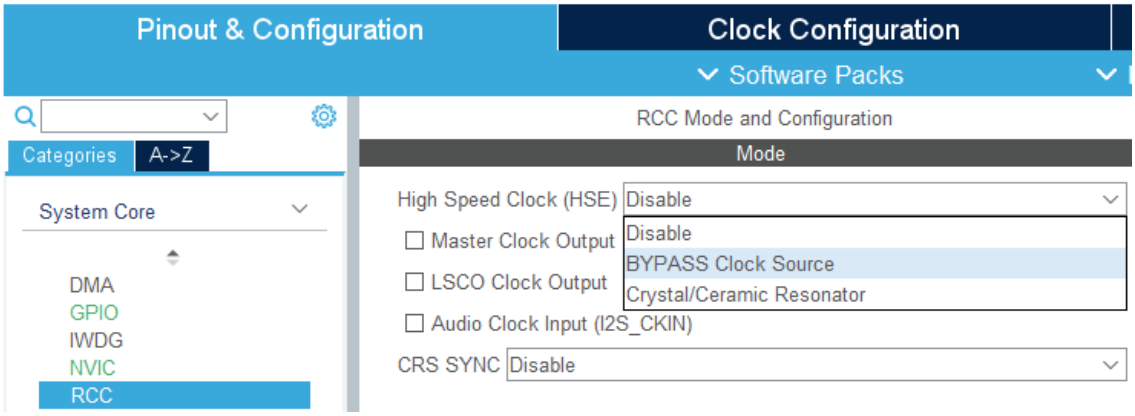

Figure 3: RCC Mode and Configuration pane and HSE choices in CubeMX.

For some details, see: https://www.youtube.com/watch?v=o4R2VNzeDhY&list=PLnMKNibPkDnGtuIl5v0CvC81Am7SKpj02&index=9&ab_channel=STMicroelectronics

Now, we can set up the exact clock rate for different parts of the MCU. We can do it by experimenting the multipliers and dividers. Or we can use automatic selection to let CubeMX to figure out the values. We prefer to use the first approach to have better control.

12 Setting up a basic timer

12.1 Choosing a timer

There is no fixed approach to choose a timer. What we use often is to choose a timer which cannot provide full functionality but is sufficient for our special application.

Here is an example. For G431n32, we can see TIM4 (timer 4) and TIM8 are not fully available due to the reason that some of the GPIO pins that have connections to these timers have been used by us for other purposes. For example, Channel 3 of TIM4 is *yellowed* out with the following message: "Channel 3 conflict with: SYS Debug Serial Wires or/and PB0-BOOT0 mapped with GPIO_Output". Well, since we only want to choose a timer for internal usages, TIM4 can be a great choice for us. This way, we can save other timers for other functionalities.

To choose TIM4, we need to go to the **Categories** tab in the left pane of the **Pinout & Configuration** tab, open up the **Timers** node, and click on **[TIM4]**.

12.2 Detailed setups of a timer

Timers can be different. Here we use TIM4 of G431n32 as an example to illustrate the setups.

12.2.1 Setups in the Mode section of TIM4 Mode and Configuration pane

Below are the **Mode** setups in the **Mode** section of the **TIM4 Mode and Configuration** pane:

- **Slave Mode**: *Disable*

- **Trigger Source**: *Disable*
- **Clock Source**: *Internal Clock* (or **Internal Clock**: *Check*)
- **Channel 1 to 4**: *Disable*
- **Combined Channels**: *Disable*
- Default choice for other selections.

Note that we will discuss channels in the setup for PWM.

12.2.2 Setups in the Configuration section of TIM4 Mode and Configuration pane

A. Setups in the **NVIC Settings** tab:

- Make sure **TIM4 global interrupt** is **Enabled**. No need to change the Preemption priority and the Sub-priority for many applications.

B. **Counter Settings** in the **Parameter Settings** tab (for an input clock of 100 MHz):

- **Prescaler**: *10000-1*
- **Counter Mode**: *Up*
- **ARR**: *5000-1*
- **Internal Clock Division**: *No Division*
- **ARR Preload**: *Disable*

C. **Trigger Output** parameters in the **Parameter Settings** tab:

- **Master/Slave Mode**: *Disable*
- **Trigger Even Selection**: *Reset*

12.3 Functions used to operate timers

Assume `HTIM_LED4` is the handler of the timer used to control LED4. We can use the following functions, defined in `stm32g44_hal_tim.c`, to start the timer:

```
HAL_TIM_Base_Start_IT(&HTIM_LED4);
HAL_TIM_Base_Start(&HTIM_LED4);
```

Note that the first one is used for starting the timer with interrupt enabled and the second for normal applications of the timer without interrupt.

In the demo project, we use the first one. Later, when we discuss assembly programming, we will use the second one to count the number of clocks used for executing specific assembly functions to evaluate their computational performance.

To set the PSC and ARR values, we can use the following macros defined in `stm32g44_hal_tim.h`.

```
__HAL_TIM_SET_PRESCALER(&HTIM_LED4, PSC_value);
__HAL_TIM_SET_AUTORELOAD(&HTIM_LED4, ARR_value);
```

Note that there are a lot of other similar definitions in this file which can be used to set up other registers of the timer.

13 Setting up PWM

13.1 Choosing a timer for PWM

Since we need to use PWM to drive the LED or other devices directly, sometimes, we don't have much choices as we do for an internal timer for interrupt only. Here, we consider the driving of the onboard LED of G431n32.

When we click on PB8 to show the menu of pin setup, we can see that this pin is connected to a number of different channels of different timers. Here, we can use TIM4_CH3 for PB8. To do so, we just need to click *TIM4_CH3* in the menu of pin setup. We should be able to see that the pin is displayed in yellow, which means we need more setups. Before moving on, let's change the label of this pin to LED4.

We need to follow the steps below to set up TIM4 for PWM output. We need to go to the **Categories** tab in the left pane of the **Pinout & Configuration** tab, open up the **Timers** node, and click on **[TIM4]**.

13.2 Detailed setups of a timer for PWM

PWM setup for each timer can be different. Here we use TIM4 of G431n32 as an example to illustrate the setups.

13.2.1 Setups in the Mode section of TIM4 Mode and Configuration pane

Mode setups in the **Mode** section of the **TIM4 Mode and Configuration** pane:

- **Slave Mode**: *Disable*
- **Trigger Source**: *Disable*
- **Clock Source**: *Internal Clock* (or **Internal Clock**: *Check*)
- **Channel3:** *PMW Generation CH3*
- **Channels 1, 2, and 4**: *Disable*
- **Combined Channels**: *Disable*
- Default choice for other selections.

13.2.2 Setups in the Configuration section of TIM4 Mode and Configuration pane

A. Setups in the **NVIC Settings** tab:

- The main reason we want to use PWM instead of a normal timer is that we want to avoid the interrupt from the timer. As such, we need to set the **TIM4 global interrupt** to *Disabled*.

B. **Counter Settings** in the **Parameter Settings** tab (for an input clock of 100 MHz):

- **Prescaler**: *10000-1*
- **Counter Mode**: *Up*
- **ARR**: *5000-1*
- **Internal Clock Division**: *No Division*
- **ARR Preload**: *Disable*

C. **Trigger Output** parameters in the **Parameter Settings** tab:

- **Master/Slave Mode**: *Disable*
- **Trigger Even Selection**: *Reset*

D. **Clear Input** in the **Parameter Settings** tab:

- **Clear Input Source**: *Disable*.

E. **PWM Generation Channel 3** in the **Parameter Settings** tab:

- **Mode**: *PWM mode 1*. Calculation for duty cycle in PWM mode 1 is straightforward.
- **Pulse (16 bits value)**: *2500-1*.
- **Output compare preload**: *Enable*.
- **Fast Mode**: *Enable*.
- **CH Polarity**: *High*.

13.3 Functions used to operate timers

Assume `HTIM_LED4` is the handler of TIM4 used to drive LED4 and `TIM_CHANNEL_LED4` is channel 3 used to directly drive LED4. We can use the following functions, defined in `stm32g44_hal_tim.c`, to start the timer:

```
HAL_TIM_PWM_Start(&HTIM_LED4, TIM_CHANNEL_LED4);
```

The other functions used in the project are used for controlling the PSC, ARR, and CCR registers. They are listed below as wrapper functions for easy use.

```
void mp_set_timer_prescaler(uint16_t PSC_value) {
    __HAL_TIM_SET_PRESCALER(&HTIM_LED4, PSC_value);
}

void mp_set_timer_autoreload(uint16_t ARR_value) {
    __HAL_TIM_SET_AUTORELOAD(&HTIM_LED4, ARR_value);
}

void mp_set_timer_compare(uint16_t CCR_value) {
    __HAL_TIM_SET_COMPARE(&HTIM_LED4, TIM_CHANNEL_LED4, CCR_value);
}
```

14 Setting up for using native FreeRTOS

We can easily configure the project in CubeMX to use FreeRTOS. Yet, if we use this route, we have to use the CMSIS API for FreeRTOS.

We will not use this approach as we prefer to use the native FreeRTOS API. To this end, we need to set up the project using an alternative approach, as discussed below.

The basic idea is to add the source files of FreeRTOS in the same way as adding other library files that we have to do manually. This will be ok as loge as we don't have any conflict between the generated code and the FreeRTOS library code. To avoid the conflicts, we need to do the following:

- Change the **Timebase Source** from the default SysTick timer to a general-purpose timer.

- Remove the ISRs for the following interrupts as they are provided in FreeRTOS:
  - System service call via the SWI instruction.
  - Pendable request for system service.
  - SysTick timer.

14.1 Changing the Timebase Source

- Go to the **Categories** tab in the left pane of the **Pinout & Configuration** tab.
- Open up the **System Core** node and click on **[SYS]**. Now, we should see a **SYS Mode and Configuration** pane to the right of the pane where the **Categories** tab is displayed. In this pane, we will see a single **Mode** group.
- Under the **Timebase Source** dropdown, select any timer that is available, say TIM6.

With this setup, we will have SysTick timer reserved for FreeRTOS.

14.2 Removing the ISRs

This can be done by following the steps below:

- Go to the **Categories** tab in the left pane of the **Pinout & Configuration** tab.
- Open up the **System Core** node and click on **[NVIC]**. Now, we should see a **NVIC Mode and Configuration** pane to the right of the pane where the **Categories** tab is displayed. In this pane, we will see a single **Configuration** group, under which we will see two tabs.
- Click the **Code generation** tab and uncheck the checkbox on the **Generate IRQ handler** column for the above interrupts.

Additionally, to match up with the priority setup in FreeRTOS, we need to configure the priority group to use 4 bits for the preemptive priority, as shown below.

14.3 Setting the priority group

- Go to the **Categories** tab in the left pane of the **Pinout & Configuration** tab.
- Open up the **System Core** node and click on **[NVIC]**. Now, we should see a **NVIC Mode and Configuration** pane to the right of the pane where the **Categories** tab is displayed. In this pane, we will see a single **Configuration** group, under which we will see two tabs.
- Click the **NVIC** tab and select **4 bits for preemption priority 0 bits for subpriority** for **Priority Group**.

15 Common tricks of using CubeMX

15.1 Seeing the alternative pins

To provide most flexibility, a certain peripheral may be connected to multiple pins. When we activate a peripheral, CubeMX may suggest some pins that we don't want to use. In this case, we can ask CubeMX to suggest alternative pins. This can be done by holding both the control key and the left button of the mouse when the cursor is on the selected pin. The alternative pins will be highlighted in dark.

Note that the debug pins do not have alternatives on most MCUs.

15.2 Preventing suggestions by pinning down some pins

CubeMX can automatically suggest/choose alternative pins when there is a conflict. To prevent this change for a given design of hardware, we can "Pin down" the chosen pins so that they will not be changed for new functions. This is done by right-click the pin and choose "Signal Pinning". We can undo the "Signal Pinning" of a pin as well.

15.3 Generating pairs of C and header files.

To make the project more portable, we can generate the initialization code in pairs of .c/.h files by selecting "Generate peripheral initializations as a pair of '.c/.' files per peripheral."

15.4 Generating report

We can generate a report of the project by clicking "File -> Generate Report".

15.5 Using the Pinout menu

There is a **Pinout** menu in the **Pinout & Configuration** tab. There are a lot of choices we can do over there. At least, the **[List Pinout Compatible MCUs]** is useful when we need to see alternative MCUs without changing the PCB.