



Australian
National
University

Lab Six

Programming the STM32 microcontroller

ENGN4213/6213

Digital Systems and Microprocessors

Semester 1, 2023

Copyright © 2023, The Australian National University

TABLE OF CONTENTS

1. Introduction	3
2. Pre-lab Task.....	3
3. What You Need.....	3
4. STM32F411RE Nucleo-64 Board.....	3
4.1 Features	5
5. Development Environment – STM32CubeIDE	7
6. Activity 1: Alternate Blinking LEDs [10 marks].....	11
6.1 Compiling	11
6.2 Debugging.....	12
6.3 Adding an external LED	12
7. Activity 2: ESCAPE GAME [20 marks]	13
7.1 Morse Code LED Blinker [5 marks]	14
7.2 External Interrupt (User Push Button)	15
Appendix A: CubeMX & TrueSTUDIO.....	19
Appendix B: Internal Connection of Breadboards	20

1. Introduction

During this lab you will have the opportunity to work with a microprocessor, the STM32F411RE from STM32 family. You will learn the structures, functionalities of the board, and have hands on experience in programming and debugging.

2. Pre-lab Task

- Understand the concepts of GPIO and Interrupt on microcontrollers (Refer Chapters 5 & 6 from the book '*Programming with STM32: Getting Started with the Nucleo Board and C/C++*'. ANU students can get free access to this book at <https://learning.oreilly.com/library/view/programming-with-stm32/9781260031324/?ar>. Also attend/review Tuesday's lecture in Week 7.)
- Install STM32CubeIDE on your machine (Available from <https://www.st.com/en/development-tools/stm32cubeide.html>)
- Refresh fundamentals of C Programming (C Primer document)

3. What You Need

Hardware:

- A STM32F411RE Nucleo board
- A USB type A to mini-B cable
- Jumper wires
- A breadboard
- 3mm LEDs
- Resistors

Software:

- STM32CubeIDE Development Platform

4. STM32F411RE Nucleo-64 Board

The STM32 Nucleo-64 board allows affordable and flexible implementation of new concepts and prototypes with the STM32 microcontroller by choosing from various combinations of performance and power consumption features. The board is made of two sections, the top section being a ST-LINK/V2-1 programmer and the bottom section is the STM32 MCU. The bottom section is also fitted with two sets of connectors: the ARDUINO® Uno V3 connectivity support and the ST morpho headers that can facilitate easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields. The STM32 Nucleo-64 board does not require any separate probe as it integrates the ST-LINK debugger/programmer. The STM32 Nucleo-64 boards come with free software libraries and examples available with the STM32Cube MCU Package.



Figure 1: Top view of STM32F411RE Nucleo-64 board

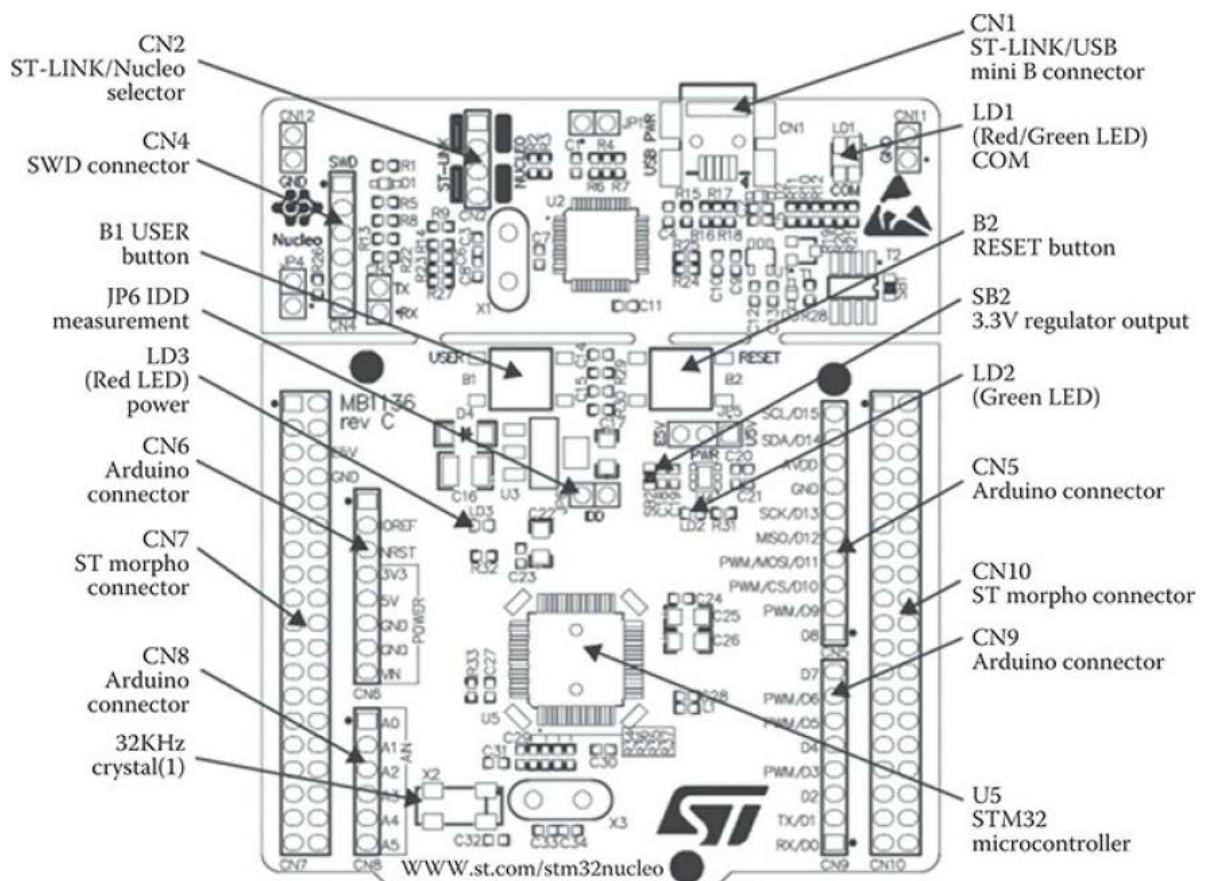


Figure 2: Annotated top view of Nucleo-64 board

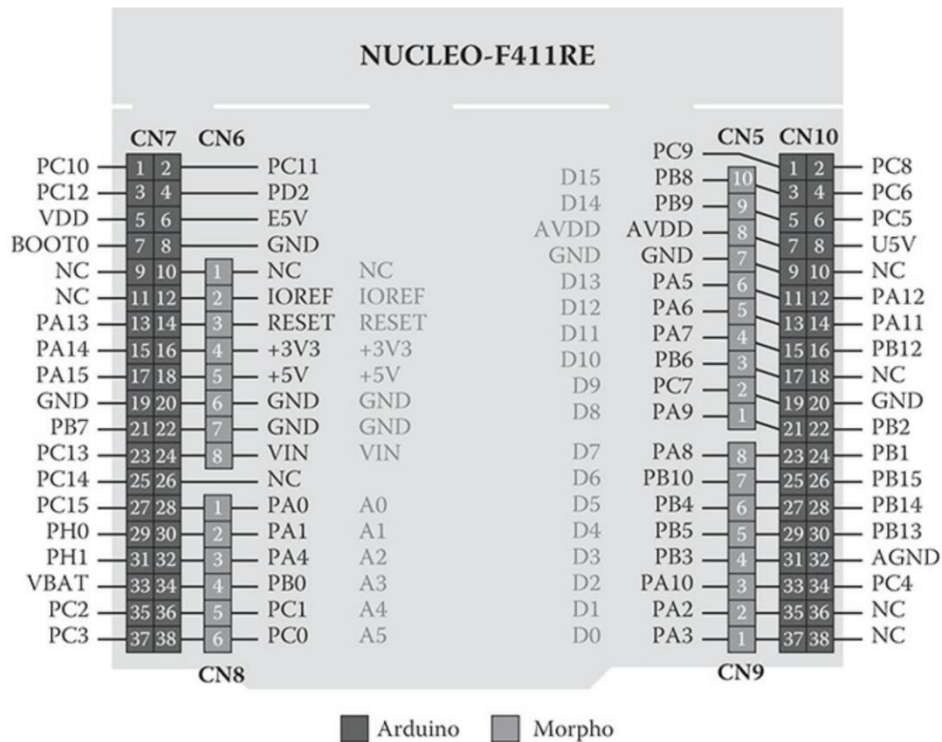


Figure 3: STM32F411RE Arduino/Morpho pin connectors

4.1 Features

Common features

- STM32 Arm®-based microcontroller in LQFP64 package
- 1 user LED shared with ARDUINO®
- 1 user and 1 reset push-buttons
- 32.768 kHz crystal oscillator
- Board connectors
 - o ARDUINO® Uno V3 expansion connector
 - o ST Morpho extension pin headers for full access to all STM32 I/Os
- Flexible power-supply options: ST-LINK, USB VBUS, or external sources
- On-board ST-LINK debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port and Debug port

Board-specific features

- External switched-mode power supply (SMPS) to generate Vcore logic supply
- 24 MHz High Speed External (HSE) Clock
- Board connectors
 - o External SMPS experimentation dedicated connector
 - o Micro-AB or Mini-AB USB connector for the ST-LINK
 - o MIPI® debug connector

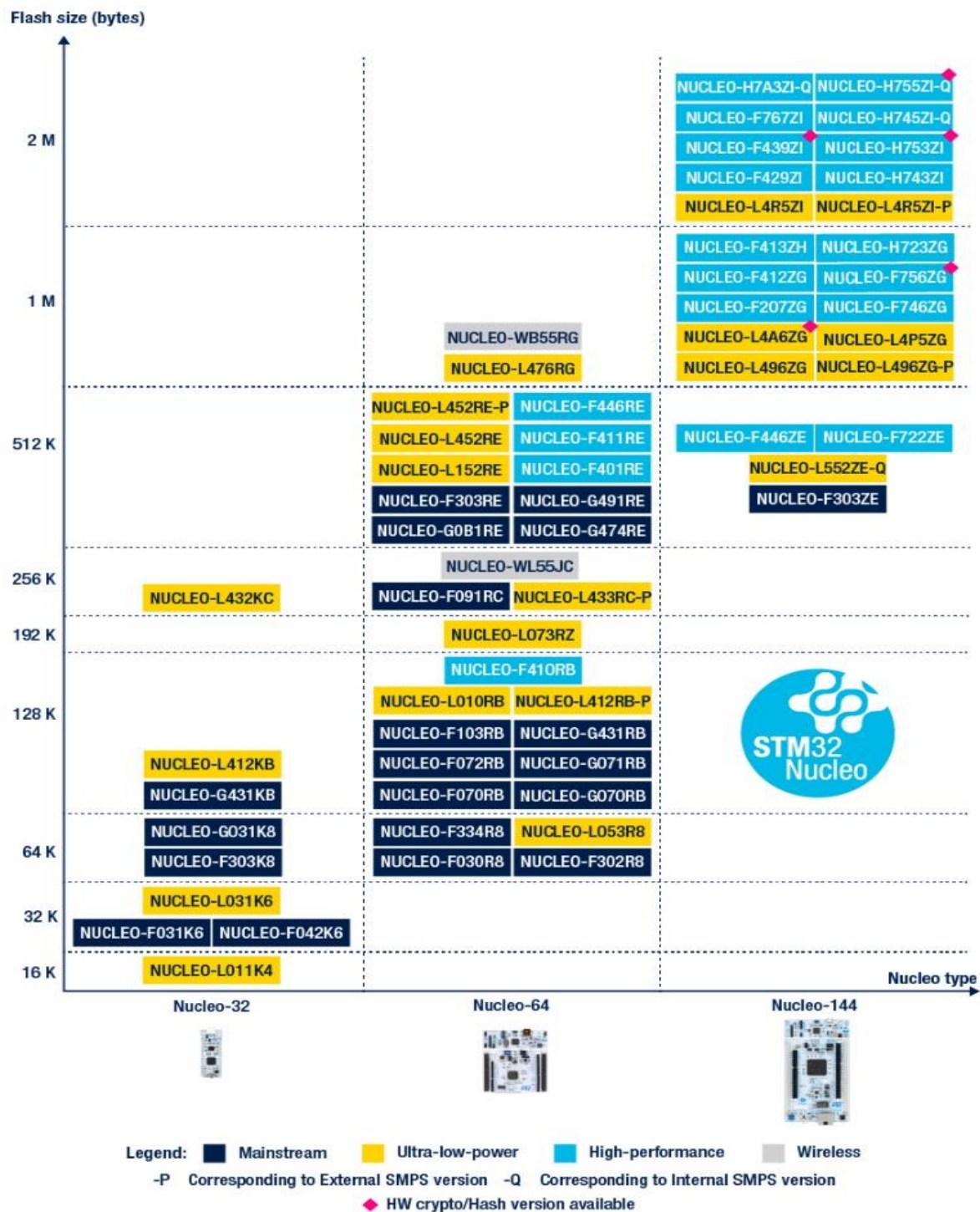


Figure 4: Available boards from STM32 Nucleo family

Apart from the F411RE board, there are other variety of boards from the STM32 Nucleo family, as shown in Figure 4. For your future projects, you can choose the suitable board according to performance specifications (pin numbers, flash size, etc.). The Nucleo boards are not the only options. You can choose any chip (not limited to STM32 ones) to design your own peripheral circuits to add more flexibility.

5. Development Environment – STM32CubeIDE

STM32CubeIDE (Available from <https://www.st.com/en/development-tools/stm32cubeide.html>) is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. Instead of STM32CubeIDE, we can also use a combination of STM32CubeMX and some other compilers (for e.g., TrueSTUDIO). If you like to use STM32CubeMX with TrueSTUDIO, please follow the instructions in Appendix A.

1. Open STM32CubeIDE. In the launcher window, you could either change the workspace address or leave as default and click on **Launch**.
2. Click on File -> New -> **STM32 Project**.
3. A target selection window should pop up. You can select from any tab on the top to look for your microcontroller or board. Since we are using a Nucleo board, we can select from **Board Selector** directly. Type “**NUCLEO-F411RE**” into the **Commercial Part Number**, the exact board should turn up in the list. **Select** the board, and click on **Next**.

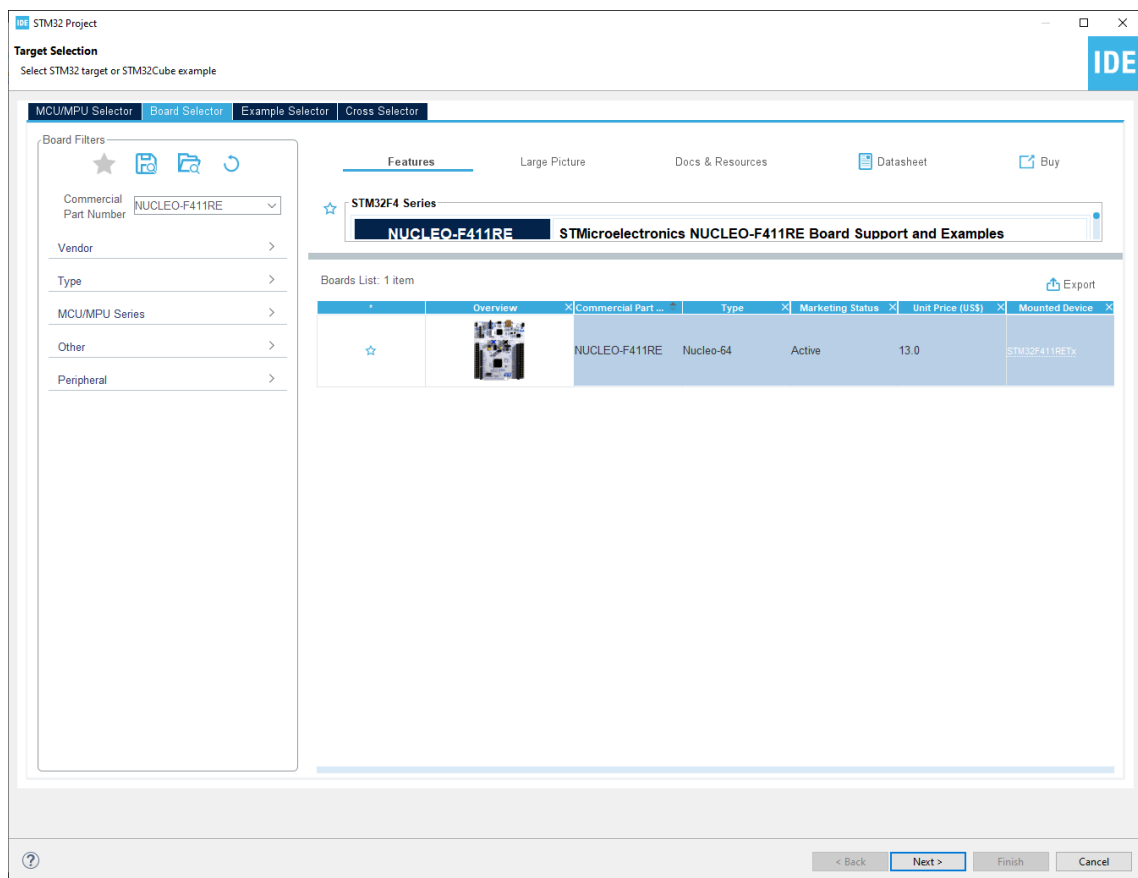


Figure 5: Board selection in STM32CubeIDE

4. A window should pop up to let you create a STM32 project. As shown in Figure 6, Give your project an identifiable **name**, select a **location** you prefer to save your project, and choose either C or C++ for the Targeted Language (the instruction is given in **C**), leave the following settings as default, and click on **Finish**.

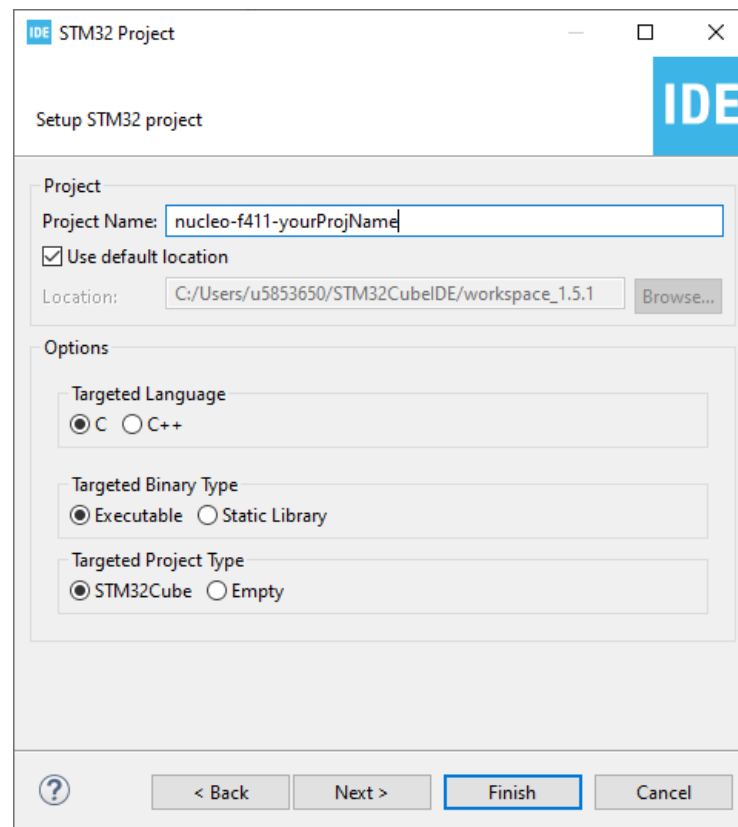


Figure 6: Creating a project

5. You can click on **Yes** if it asks to '*Initialize all peripherals with their default Mode?*'. It will give you a ready-to-go configuration of clock, connectivity, and necessary pins. You can change them later as required.
6. A pinout view of "**Pinout & Configuration**" should pop up (Figure 7). You can modify the pins according to your need, by clicking on it. The grey pins are not set. When you set a pin, it will turn green. Leave everything as default at the moment. You can also **enter a user label** for any pin by right clicking on the pin (Figure 8).

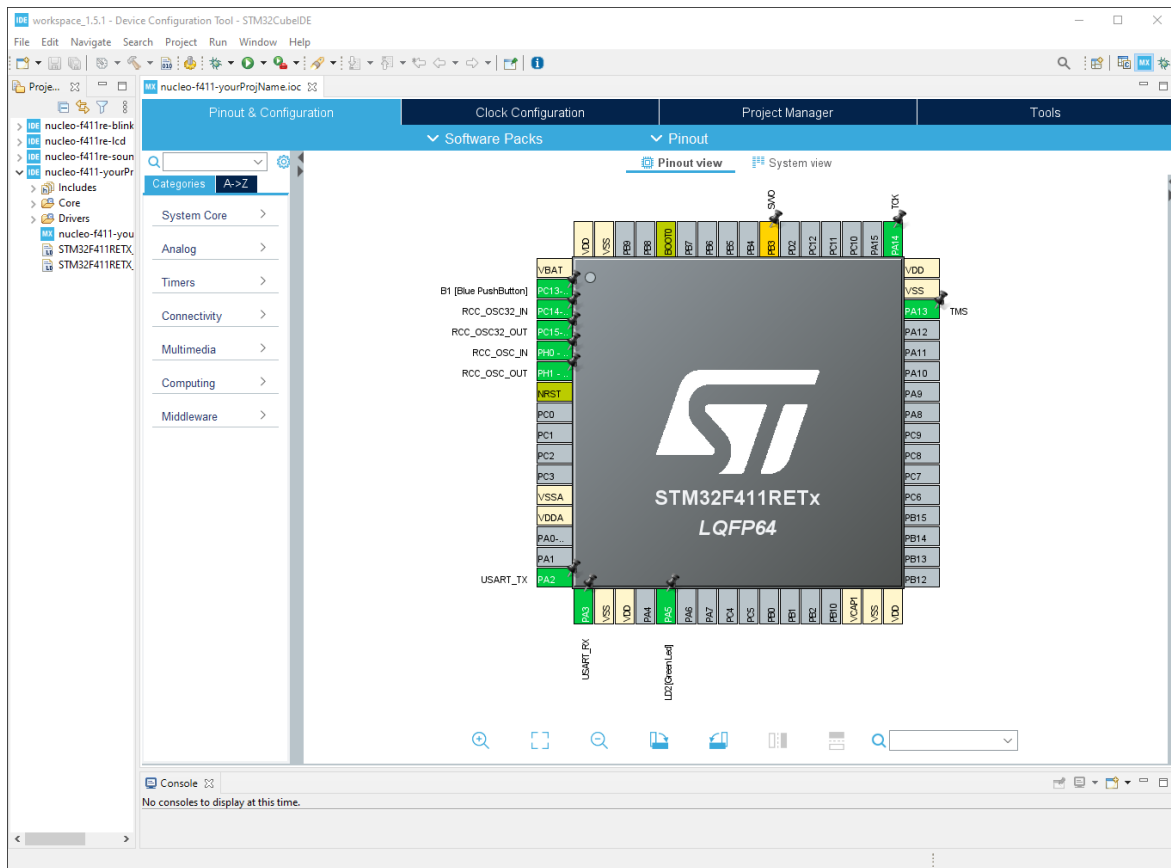


Figure 7: Pinout & Configuration view in STM32CubeIDE

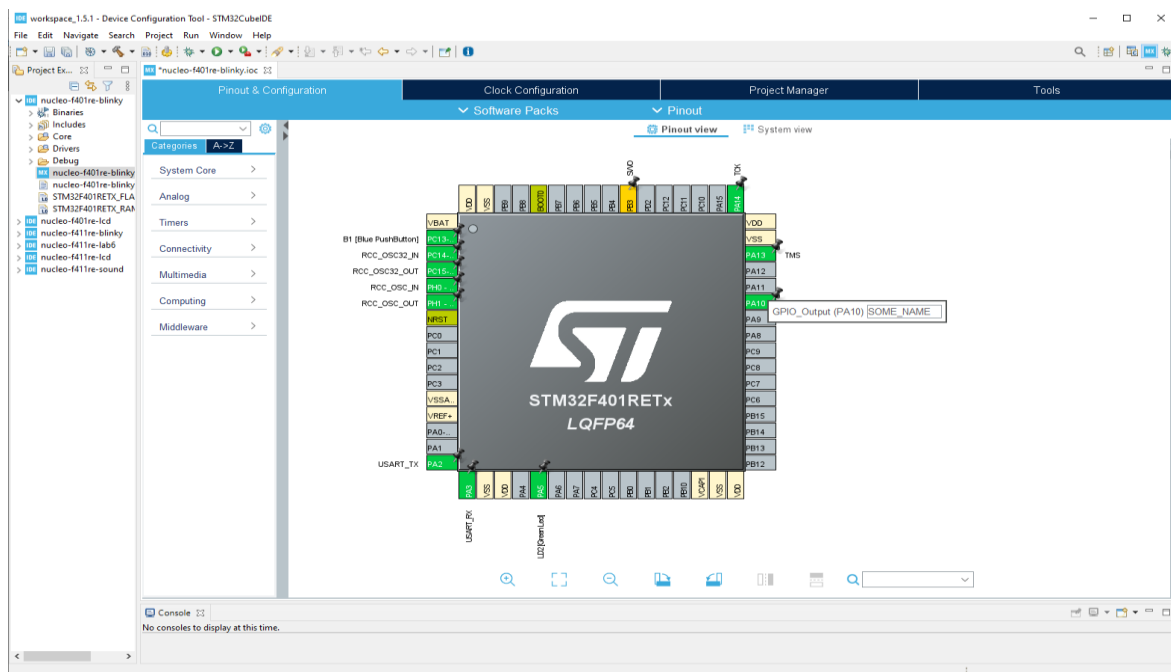


Figure 8: Enter User Label for GPIO Pin

7. If you need to change **clock configuration**, simply click on the second tab on the top. You can make changes depending on your project demand, by changing the number and fractions in the clickable boxes. Leave the configuration as default at the moment.

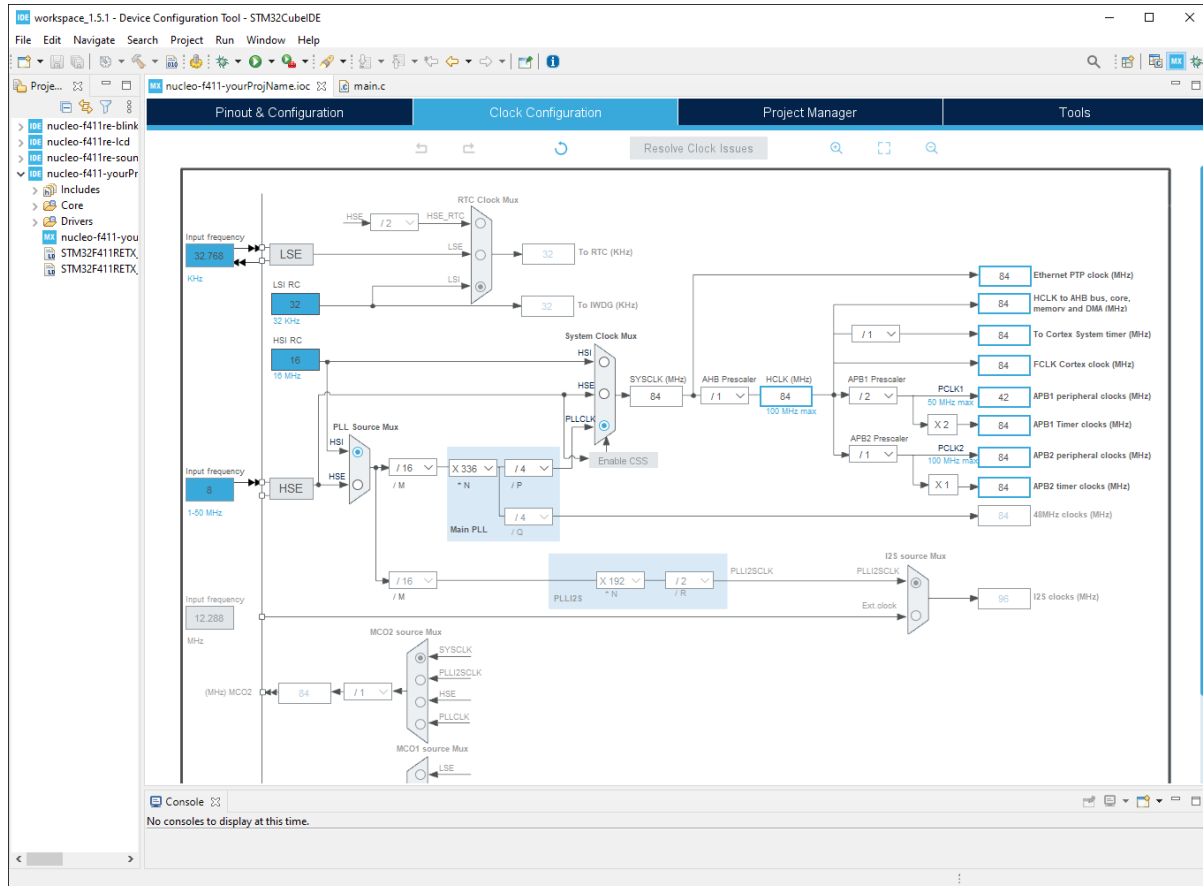



Figure 9: Clock Configuration in STM32CubeIDE

8. When you are happy with the pinout and clock settings, remember to save first and click on **Project -> Generate Code**.
9. It brings you to the compiler window, where you can write C code to program the board.
10. More information can be accessed in “**STM32CubeIDE User Guide**” in the MC Lab resources folder.

6. Activity 1: Alternate Blinking LEDs [10 marks]

In this activity, you will program STM32F4 Nucleo board to blink two LEDs alternatively.

6.1 Compiling

1. Create a new project by following the steps given in Section 5.
2. Check the “**Pinout & Configuration**” (Figure 7) to set the internal LED LD2 (green LED on the Nucleo board) to a pin. You can simply **click** on the pin and set it to “**GPIO-output**” from the options. In the default configuration shown in Figure 7, we can see that LD2 is already connected to PA5 and configured as GPIO-output. The name (user label) of a pin could be different from the pin name in the pinout view. Check **Figure 3** and “**main.h**” (inside Core->Inc subfolder) in your project folder to **confirm the pin names** before using in any function.
3. Check the HAL document UM1725 (‘stm32f4-hal’ in MC Lab resources folder) of STM32F4 series about **GPIO TOGGLEPIN functions** (Chapter 31) and **delay function** (Chapter 6).
4. Use the information you collected from steps 2 and 3 to write some code in “**main.c**” to make the internal LED blink every second. Make sure to write your code between the template comments `/* USER CODE BEGIN X */` and `/* USER CODE END X */`, where X is a number (you will find a few in “**main.c**”), or the code will disappear after you update the pinout or clock configuration.
5. Once you have done coding, click on the hammer button  on the top tool bar to build the project. There should be some status information in the Console window. If no error, it is ready for hardware debugging.

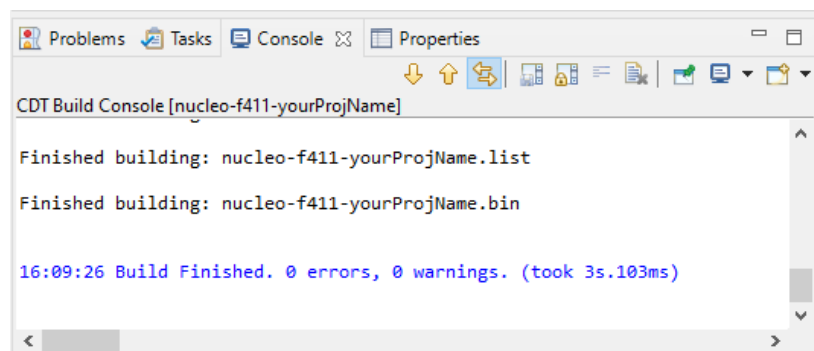




Figure 10: Build console outputs

6.2 Debugging

1. Once the build finished, you can connect the board with your PC using USB cable. If the build is successful with no errors, it is time to load your code to the Nucleo board.
2. Click on the **debug** button  on the top tool bar to run debug on the board. If a window pops up asking to **update ST-Link**, click on **Yes**. In the first attempt, the “**Debug Configurations**” window may also pop up. There simply choose **ST-LINK S/N** option under **Debugger** tab as your interface. When the debugger is connected (**Debug Perspective** is opened), you can click on **run**  and see if your LED works.
3. If you find the board does not work as you expect, try adding break points looking into values to locate the problem.

6.3 Adding an external LED

1. Connect an external LED **in series with a resistor** (to avoid thermal drift on LED) to one of the GPIOs on the Nucleo board using breadboard and several jumper wires. One way to connect LED is shown in Figure 11 – the anode of LED to GPIO via a resistor, and the cathode to ground. If you are not familiar with breadboards, check Appendix B for understanding the internal connection.



Figure 11: Possible circuit to connect a LED to a STM32 board

Hints:

- a. The typical output voltage at GPIO is 3.3V.
 - b. For LEDs, normally 5mA to 10mA current passing through is adequate.
 - c. In your lab part kit, there are 10 resistors of 560 ohms each. Feel free to use them.
 - d. A LED has two legs. Longer leg is positive leg, and shorter leg is negative leg.
 - e. Please avoid using complex connections and short circuits.
2. You can choose any of the available GPIO pins to build connection. Check the **pin connectors (Figure 3)** and your “**Pinout & Configuration**” (**Figure 7**) again.
 3. Like what you did to the internal LED in the previous task (5.1.2), make changes in the “**Pinout & Configuration**” to connect the external LED to the GPIO in your project. After setting as GPIO output, right-click on the pin and enter a user label. Save your configuration every time you make changes and generate code.

4. Modify your code to let the internal LED and external LED **blink alternatively**.
5. Successfully demonstrate your work on hardware.

7. Activity 2: ESCAPE GAME [20 marks]

-> You fainted when doing this lab.

-> When you woke up, you found yourself surrounded by darkness.

-> You tried to move around, but unable to stretch your limbs.

-> You then realized you are trapped.

-> Trapped in the STM32 CHIP on your Nucleo board.

->

-> . . .

-> You tried to contact something outside, but every attempt was swallowed up by the darkness.

-> . . .

-> However, you suddenly noticed that you could control the CHIP and its peripherals.

-> You came up with an idea - **using the embedded LED to seek for attention**.

-> A long shot, but worth a try.

7.1 Morse Code LED Blinker [5 marks]

Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes. The Morse code machine and the encoding look-up chart are shown in the following figures. Note that dots and dashes can be considered as time duration of a signal.

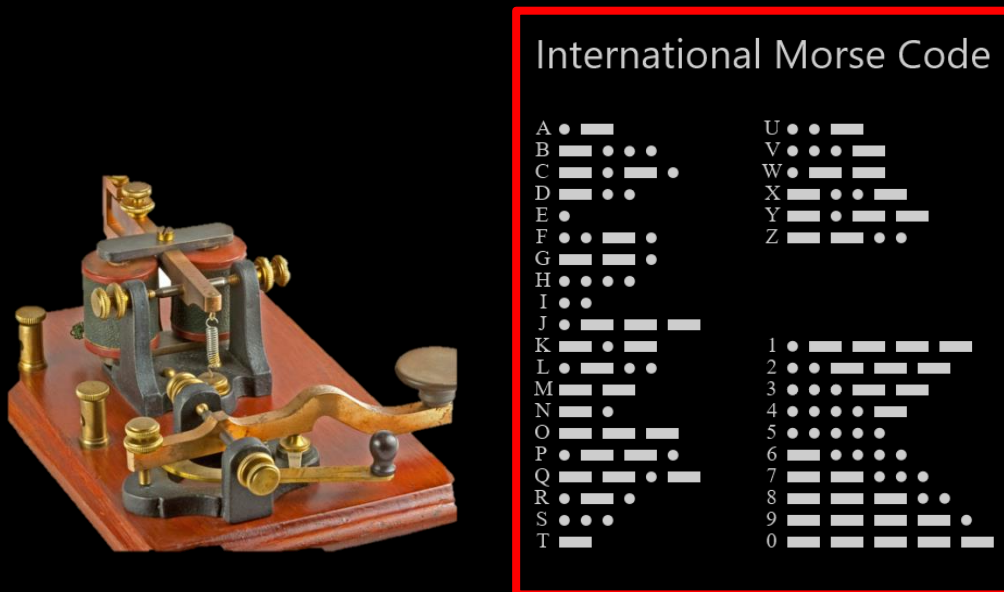


Figure 12: Morse code machine and encoding table for all letters and numbers

Try to encode a SOS message using Morse code encoding table from Figure 12 and convey that message to outside using a blinking LED.

Hints:

- The SOS message (could be anything) should be presented repeatedly when the board is powered (not just once). Note that the message can be hard-coded for this task.
- You can write a simple LED blinker function with a delay argument and then call this function repeatedly with appropriate delays to display dot/dash. It is advisable to write user-defined functions between the template comments `/* USER CODE BEGIN 4 */` and `/* USER CODE END 4 */`.
- Mind the time gap between each letter in your words, or people should find it hard to understand your message. Presenting your message in a neat manner can save your life. You can define different delay times for dot, dash, gap between dot/dash of same letter, gap between different letters and gap between words.

7.2 External Interrupt (User Push Button) [15 marks]

- > Congratulations.
- > You successfully blinked the LED in Morse code to convey your message.
- > It seems that. . . someone, or something noticed your message.
- >
- > Unfortunately, it was a bird.
- > The bird was attracted by your blinking LED and began to peck the chip.
- > You were threatened and suddenly turned off the LED.
- > The bird felt bored and left.
- > You are thinking of a flexible way to avoid the bird and convey the message.
- >
- >
- > Blink the SOS message as usual.
- > When the bird approaches, immediately stop LED blinking for 3 seconds. You can use a push button to convey the interrupt due to the bird.
- > After 3 seconds, start blinking from the beginning of the message.
- >
- > Please survive.

Instructions:

1. On your Nucleo board, set the **blue push button** to be the **external interrupt**. (Hint: the blue push button is connected to GPIO PC13. Set this pin as **GPIO_EXTI13** in “Pinout & Configuration”.)
2. In your “Pinout & Configuration” window, select **GPIO** from the left sidebar. As shown in Figure 13, click on PC13 pin in the list, then change the GPIO mode to “External Interrupt Mode with Falling edge trigger detection”, and give a recognizable name in User Label.

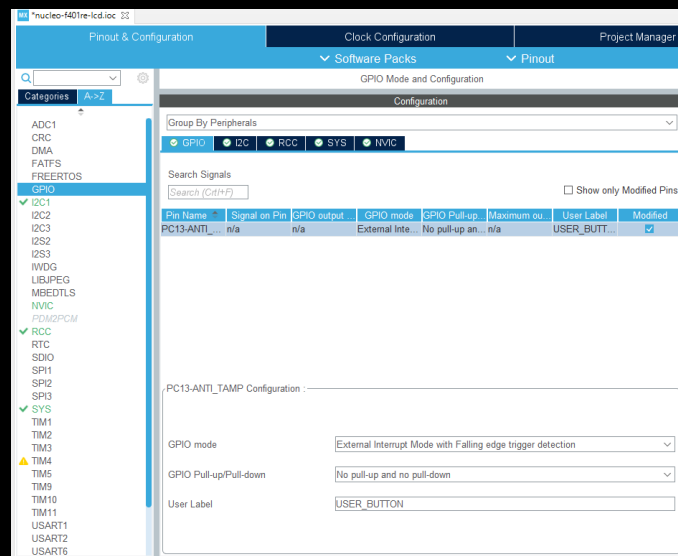


Figure 13: Modify GPIO Mode.

3. Enable the external interrupt. In the selection of GPIO, click on the NVIC tab, and enable the interrupt “EXTI line [15:10] interrupts” (Figure 14). Save your configuration and generate code.

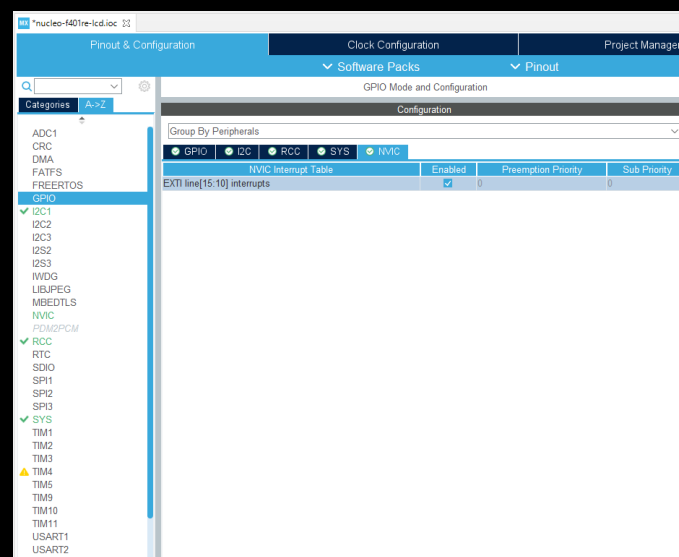


Figure 14: Enable External Interrupt

- Implement interrupt callback function to immediately stop blinking for a certain time when the user button (set as external interrupt) is pushed.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    // Write your code here
    // (Hint: You can set a global variable 'flag' to 1 to indicate that
    // the interrupt is triggered. Also, reset the blinking LED.)
}
```

This function will be executed when the interrupt is triggered and then the program returns to the instruction where it was initially interrupted.

- You might not get expected output if HAL_Delay is used inside the external interrupt callback function due to clash in their interrupt priorities (Check the HAL document for more information). If you want to use HAL_Delay in HAL_GPIO_EXTI_Callback, set a lower priority (higher numerical value) in the 'Preemption Priority' of EXTI line under NVIC configuration as shown in Figure 15. Save configuration and generate code.

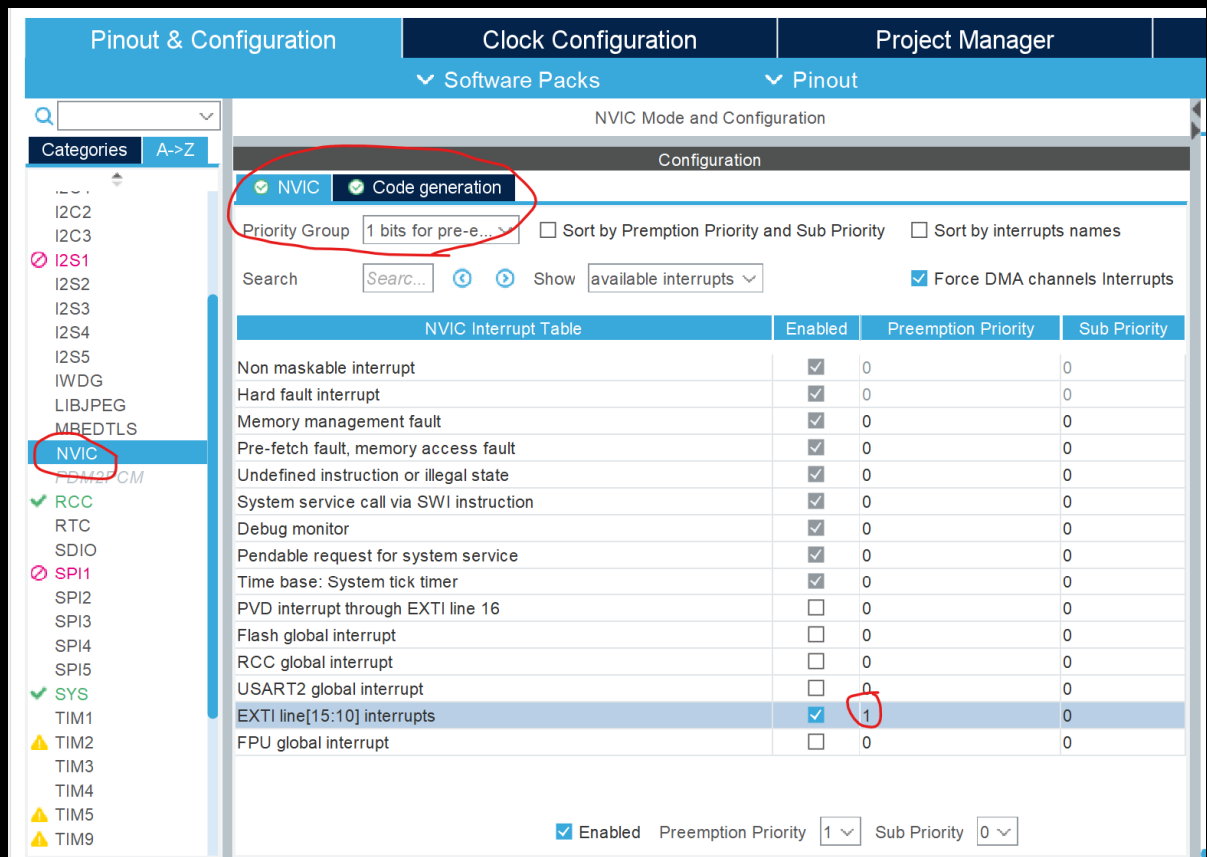


Figure 15: Priority Modification under NVIC Configuration

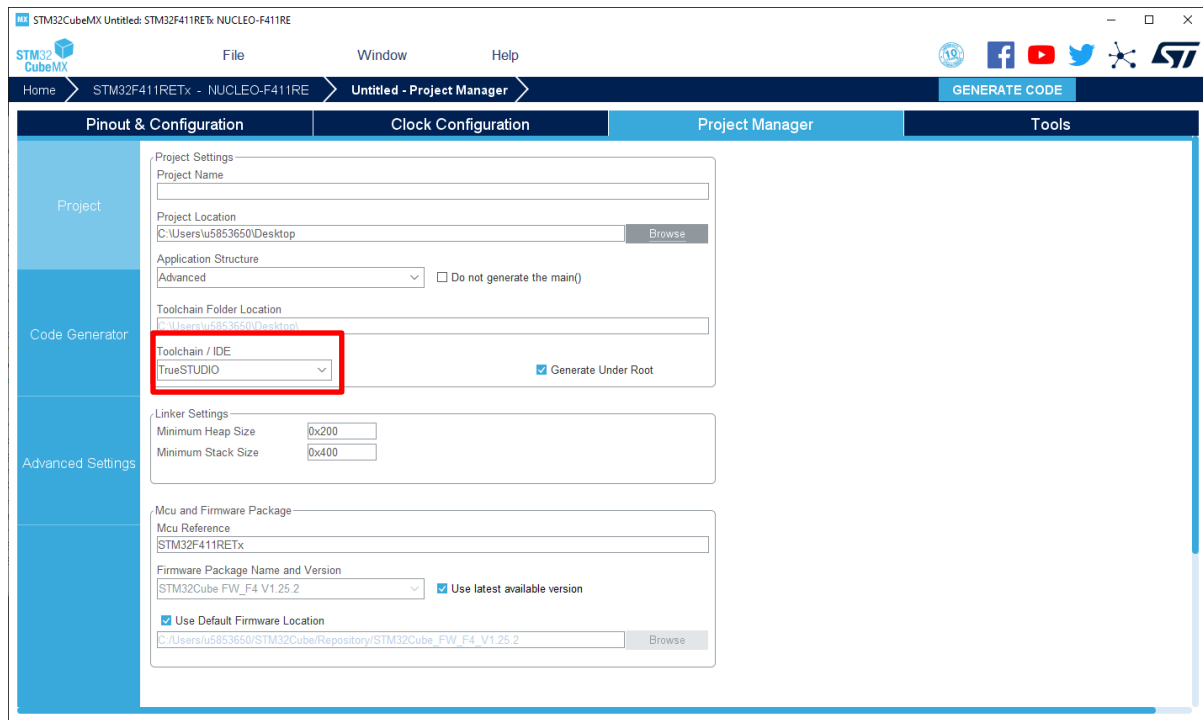
Another way is to set a global flag variable inside the external interrupt routine and monitor its condition in the main function to perform appropriate tasks: whether to show the message or pause.

Note that after the pause time, the LED should blink the message again from the beginning.

6. Successfully compile and demonstrate your work.
 - a. Whenever you push the button, the LED immediately stops blinking.
 - b. After a certain pause time, the LED resumes to blink the message from the beginning (i.e., starting from the first letter of your message).

Appendix A: CubeMX & TrueSTUDIO

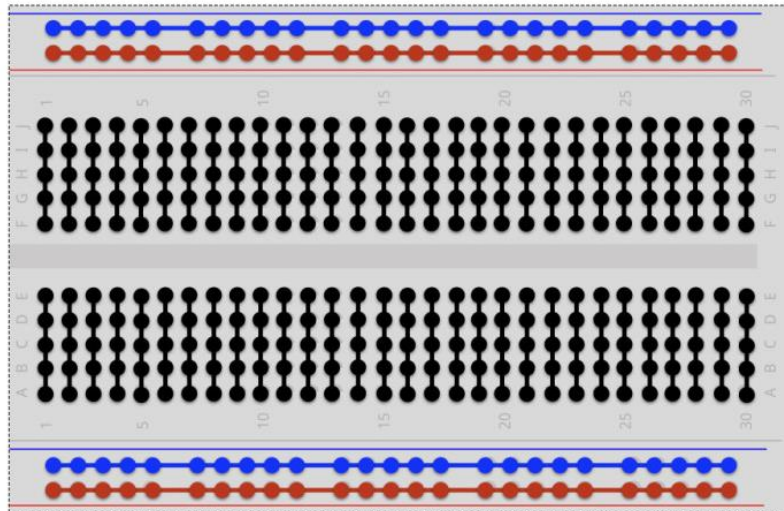
If you are not using STM32CubeIDE, but using a combination of CubeMX and some other compilers (for example TrueSTUDIO), you can simply change the **Toolchain / IDE** in **Project Manager** tab (in your CubeMX).



Then, click on **GENERATE CODE** on the right top corner, you will have a project folder.

Open TrueSTUDIO, go to File -> Open Projects from File System, import the folder from the location you just saved. Then you can do the same thing as using STM32CubeIDE.

Appendix B: Internal Connection of Breadboards



The nodes in line are connected.

(For a few boards, each red/blue line is divided into two parts.)