

LAB 02

GPIO AND LED



UNIVERSITY
OF TRENTO

Prof. Davide Brunelli

Dept. of Industrial Engineering – DII

University of Trento, Italy

davide.brunelli@unitn.it

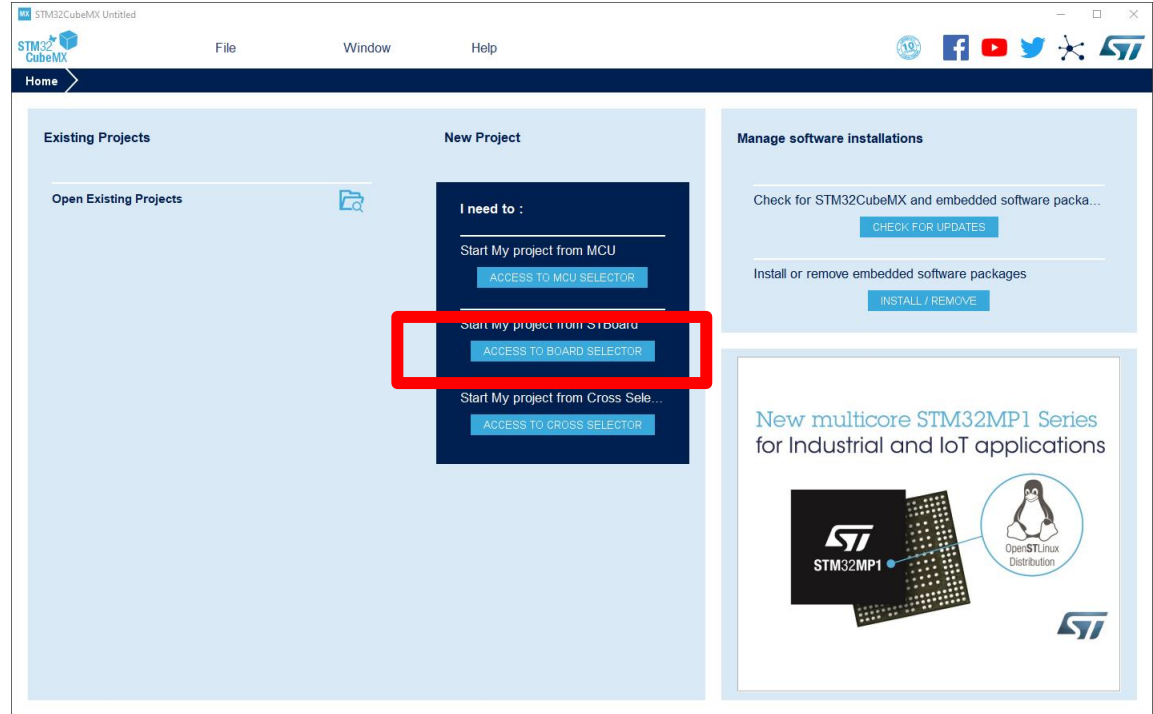


STM32 CUBEMX

STM32 CUBEMX – SELECTING THE BOARD

A graphical tool which enables **fast configuration** of STM32 microcontrollers and microprocessors.

Generates the corresponding **C code** for the Arm Cortex core.



STM32 CUBEMX – SELECTING THE BOARD

New Project from a Board

MCUMPU Selector | **Board Selector** | Example Selector | Cross Selector

Board Filters

Commercial Part Number: NUCLEO-L476RG

Type: >

MCU/MPU Series: >

Other: >

Peripheral: >



Features | Large Picture | Docs & Resources | Datasheet | Buy | Start Project

Datasheet document is not available

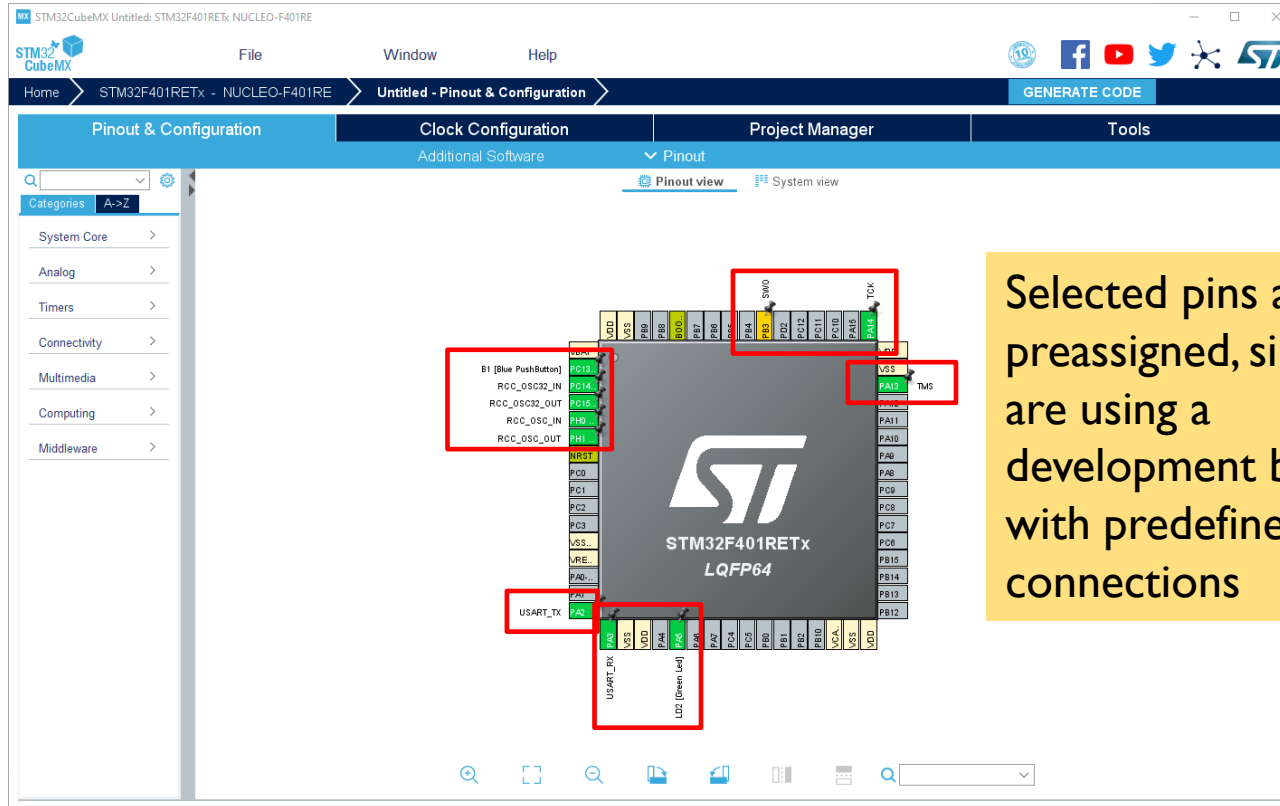
SIL Ready | ASIL Ready | ClassB Ready | Partner Program

Build your certified safety system with STM32 and STM8

Boards List: 1 item

	Overview	Commercial Part No	Type	Marketing Status	Unit Price (US\$)	Mounted Device
		NUCLEO-L476RG	Nucleo-64	Active	14.0	STM32L476RGTx

STM32 CUBEMX – MULTIPLEXING THE PINS



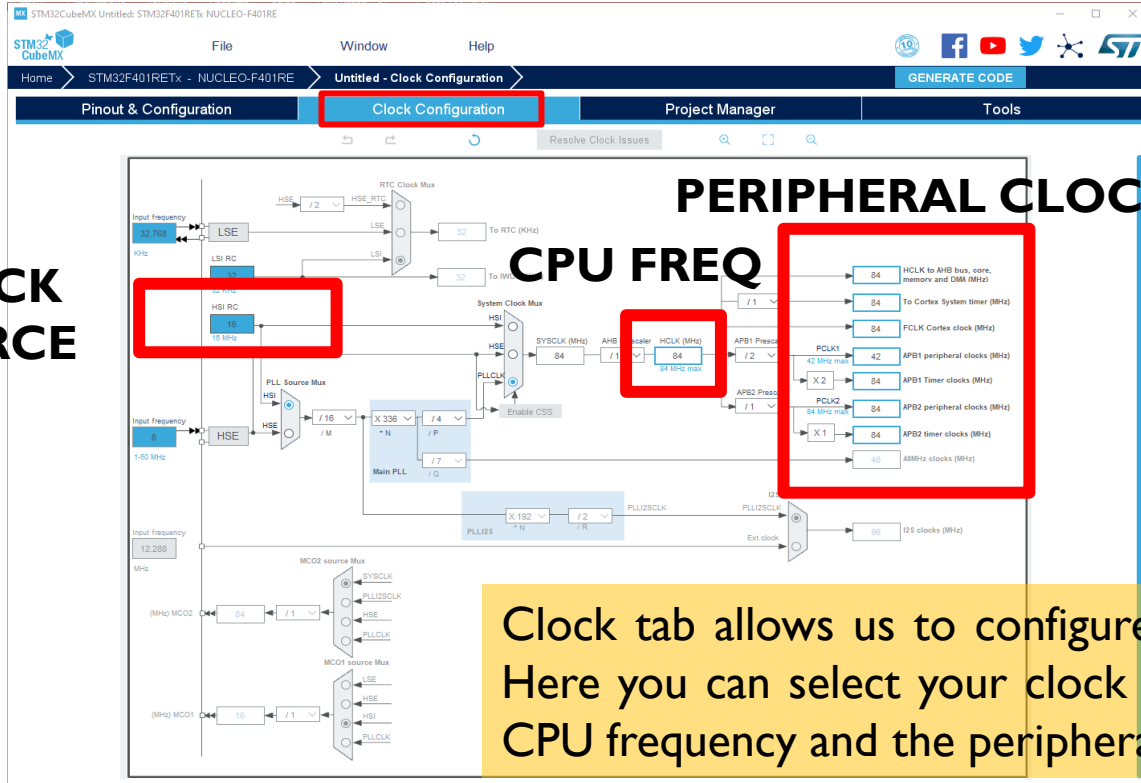
Selected pins are preassigned, since we are using a development board, with predefined connections

STM32 CUBEMX – SETTING THE CLOCKS

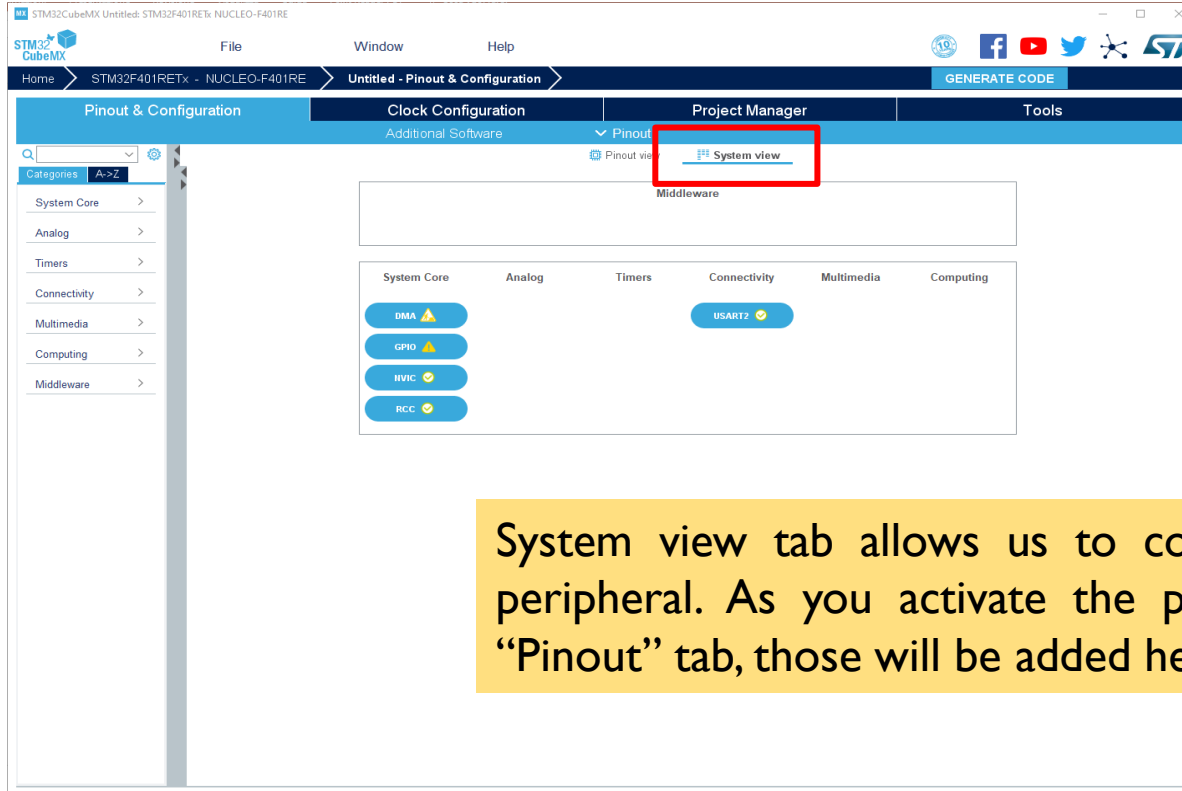
**CLOCK
SOURCE**

CPU FREQ

PERIPHERAL CLOCK

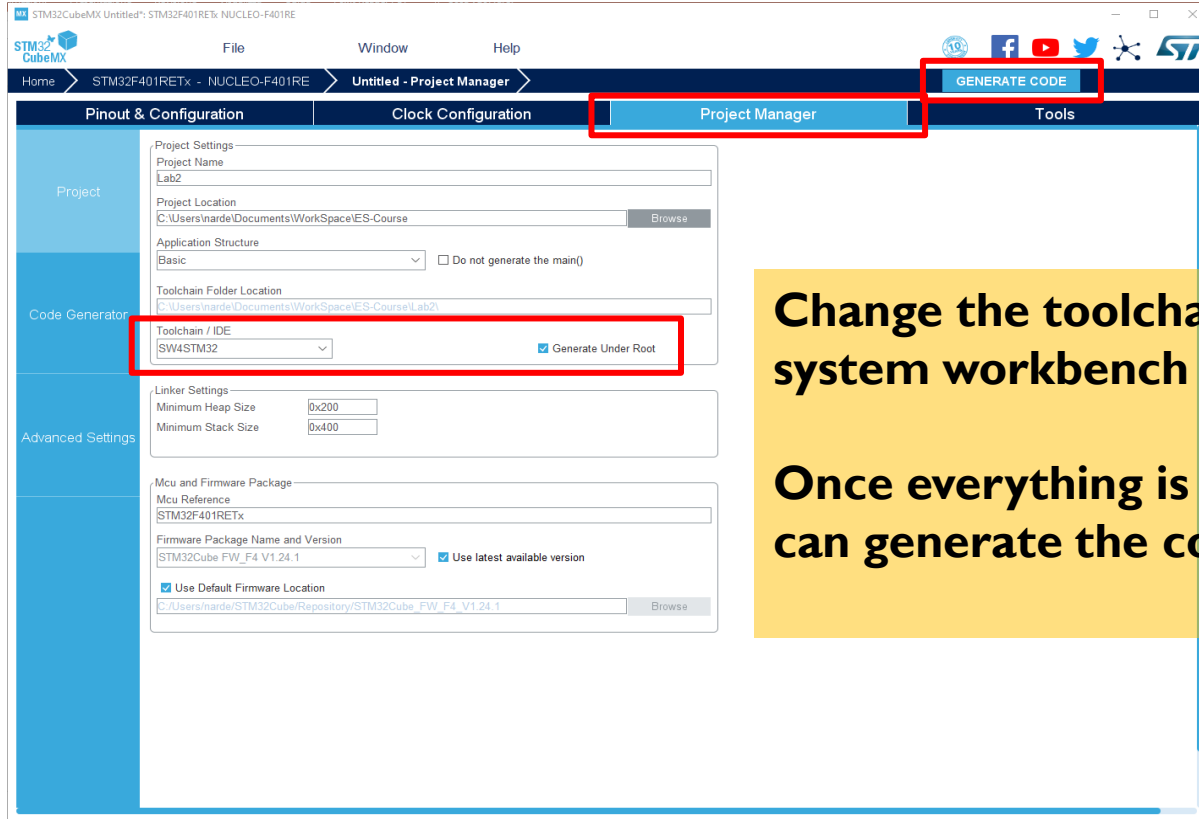


STM32 CUBEMX – CONFIGURING THE PERIPHERALS



System view tab allows us to configure all the peripheral. As you activate the peripheral from “Pinout” tab, those will be added here

STM32 CUBEMX – SETTING THE TOOLCHAIN



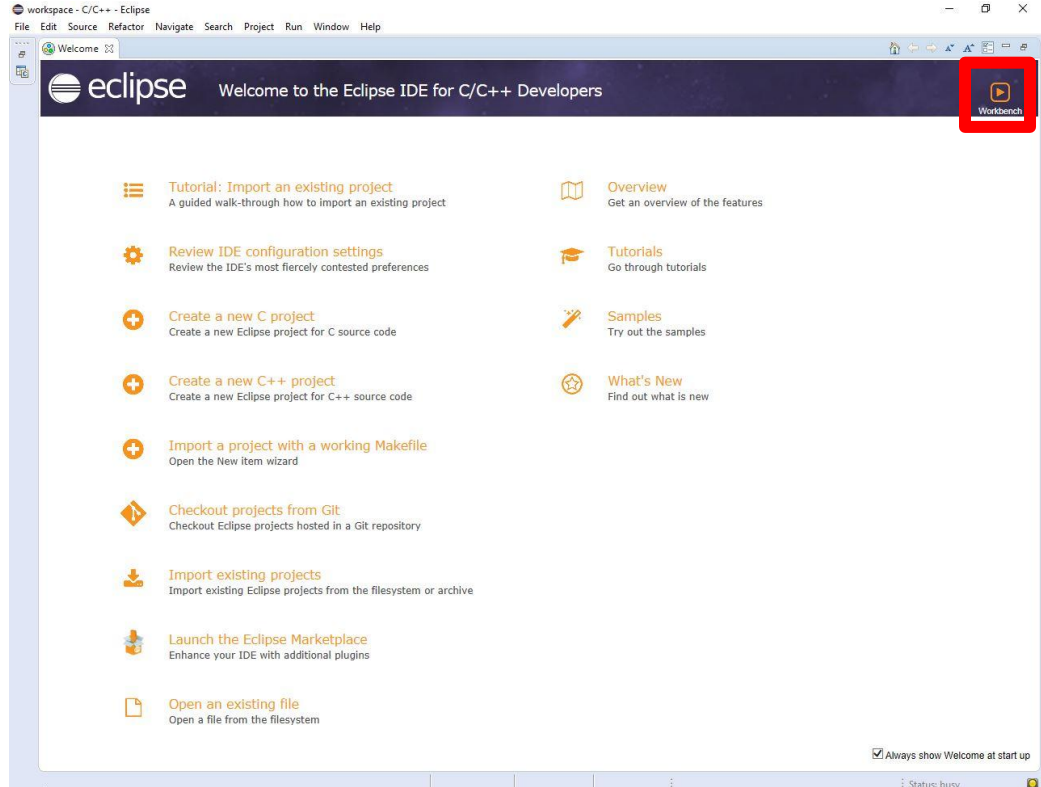
Change the toolchain / IDE to system workbench

Once everything is configure, we can generate the code



SYSTEMWORKBENCH FOR STM32

STM32 – SYSTEM WORKBENCH



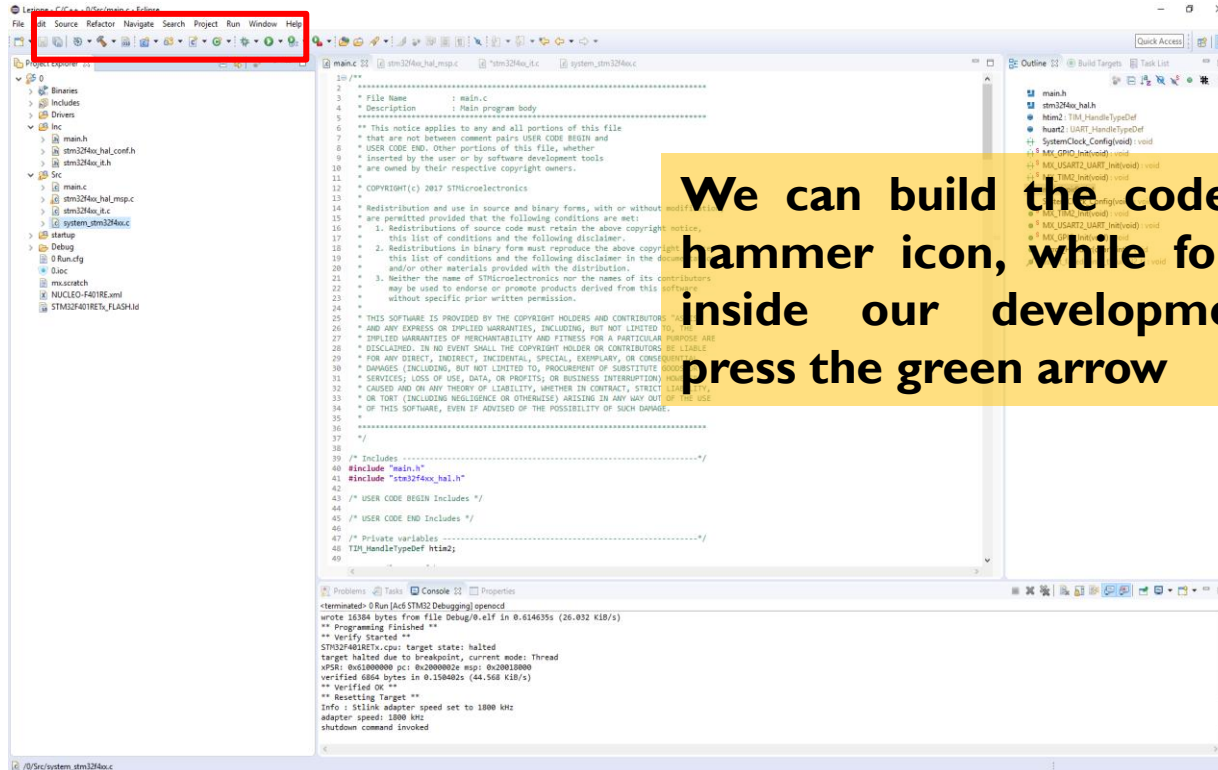
STM32 – SYSTEM WORKBENCH

The screenshot shows the STM32CubeIDE IDE interface. On the left, the 'Project Explorer' pane displays the project structure, with the 'Src' folder expanded and the file 'main.c' selected. A red rectangle highlights this area. The main editor window displays the code of 'main.c', which includes headers for 'stm32f4xx_hal.h' and 'stm32f4xx_it.h', and defines the 'main' function. The 'Outline' pane on the right shows the project structure, including 'main.c' and 'stm32f4xx_it.h'. The 'Console' pane at the bottom shows the output of the program, including the message 'Programme Finished' and the state of the target device.

main.c
stm32f4xx_it.c
stm32f4xx_hal_msp.c
system_stm32f4xx.c

-> Main file
-> Interrupt service routine
-> HAL library initialization
-> System low level config

STM32 – SYSTEM WORKBENCH



ADDITIONAL MATERIAL

- **MCU Datasheet**

<http://www.st.com/content/ccc/resource/technical/document/datasheet/30/91/86/2d/db/94/4a/d6/DM00102166.pdf/files/DM00102166.pdf/jcr:content/translations/en.DM00102166.pdf>

- **STM32 Nucleo-64 User manual**

http://www.st.com/content/ccc/resource/technical/document/user__manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf

EXERCISE 1: TOGGLE THE LED

EX I – TOGGLE THE LED

This demo is focused on **GPIO**, in particular 2 GPIO already wired to

- the **Green Led (LD2)** through **PIN 5**,
- the **Blue Button (BI)** through **PIN 13**.

GPIO (General Purpose Input Output) is the interface that allows microprocessor / memory to communicate through Input-Output channels. In our case, Input and Output are managed by two 2-state Pins (5 and 13)

We will see how to switch the led in 3 ways:

- **Polling the Blue Button**
- **By direct Read/Write on Pin using a delay**
- **Capture an Interrupt generated by the Blue Button.**

STM32 GPIO – SET/RESET

void **HAL_GPIO_TogglePin**(**GPIO_TypeDef*** GPIOx, **uint16_t** GPIO_Pin)

```
/**
 * @brief Toggles the specified GPIO pins.
 * @param GPIOx Where x can be (A..K)
 * @param GPIO Pin Specifies the pins to be toggled.
 * @retval None
 */
```

```
typedef enum
{
    GPIO_PIN_RESET = 0,
    GPIO_PIN_SET
}GPIO_PinState;
```


STM32 GPIO – SET/RESET

void **HAL_GPIO_WritePin**(**GPIO_TypeDef*** GPIOx, **uint16_t** GPIO_Pin, **GPIO_PinState** PinState)

- * @brief Sets or clears the selected data port bit.
- * @note This function uses GPIOx_BSRR register to allow atomic read/modify accesses. In this way,
* there is no risk of an IRQ occurring between the read and the modify access.
- * @param **GPIOx** where x can be (A..K)
- * @param **GPIO_Pin** specifies the port bit to be written.
* This parameter can be one of **GPIO_PIN_x** where x can be (0..15).
- * @param **PinState** specifies the value to be written to the selected bit.
* This parameter can be one of the **GPIO_PinState** enum values:
* @arg **GPIO_PIN_RESET**: to clear the port pin
* @arg **GPIO_PIN_SET**: to set the port pin
- * @retval None

STM32 GPIO - READ

GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

```
/**
 * @brief Reads the specified input port pin.
 * @param GPIOx where x can be (A..K) to select the GPIO peripheral for
STM32F429X
 * @param GPIO Pin specifies the port bit to read. This parameter can be
GPIO_PIN_x
           where x can be (0..15).
 * @retval The input port pin value.
 */
```

STM32 GPIO – TOGGLE BY INTERRUPT

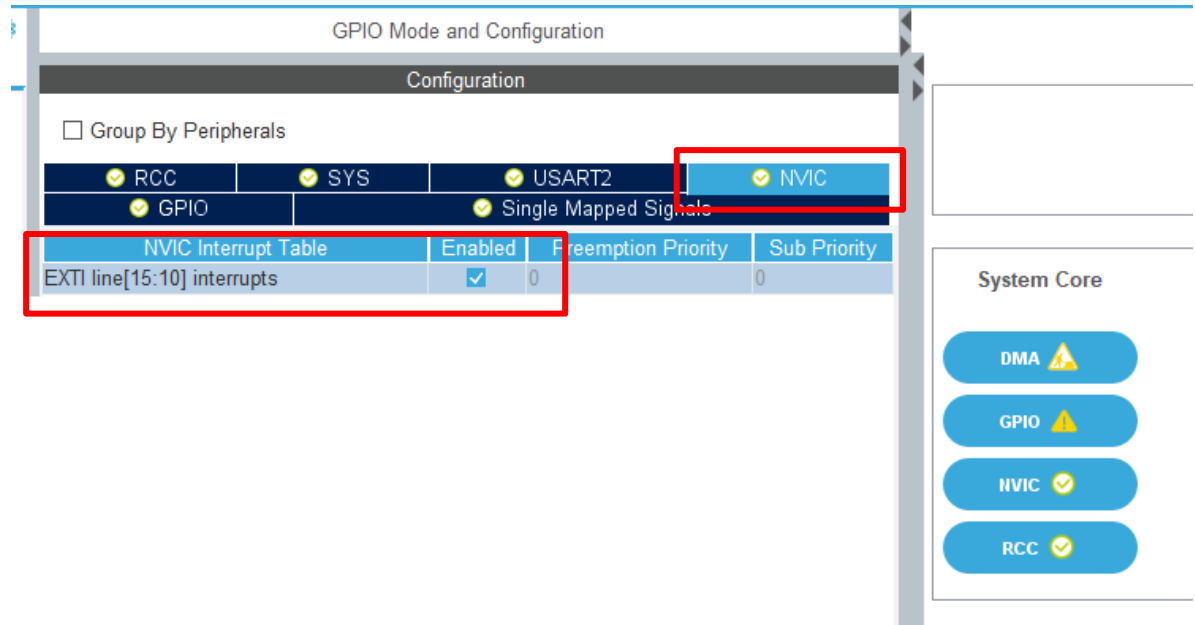
Checking the LED, you can notice that the behavior isn't always as expected: most times it toggles correctly, but sometimes the LED looks like switching ON (or OFF), erroneously. This because we can miss Blue Button press event or we can double read it.

To solve this trouble, we introduce the **Interrupt** technique. The generates a signal alerting the system of the Pin I3 activation. When the Interrupts is received, the Led toggles.

STM32 GPIO – TOGGLE BY INTERRUPT

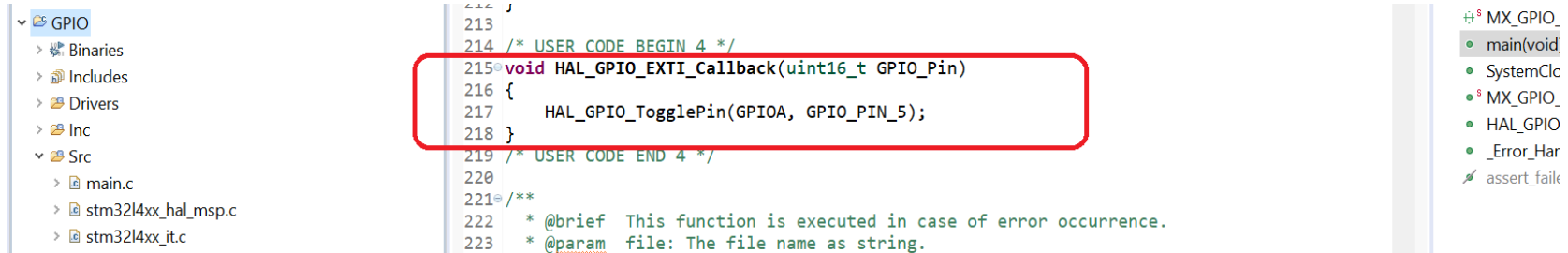
To enable the interrupt, go back to cubeMX and under **NVIC** activate Interrupts **EXTI Line [15:10]**.

Once done, re-generate the code.



STM32 GPIO – TOGGLE BY INTERRUPT

Once activated the interrupt, we have to add the callback that is called when the interrupt is fired. Inside this function, we toggle the LED status



The screenshot shows an IDE with a project tree on the left, a code editor in the center, and a symbol table on the right. The project tree shows a 'GPIO' folder containing 'Binaries', 'Includes', 'Drivers', 'Inc', and 'Src'. The 'Src' folder contains 'main.c', 'stm32l4xx_hal_msp.c', and 'stm32l4xx_it.c'. The code editor shows the implementation of the `HAL_GPIO_EXTI_Callback` function in `stm32l4xx_it.c`. The function is defined as `void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)` and contains a single line of code: `HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);`. This function call is highlighted with a red rectangle. The symbol table on the right lists the following symbols: `MX_GPIO_`, `main(void)`, `SystemClc`, `MX_GPIO_`, `HAL_GPIO`, `_Error_Har`, and `assert_fail`.

```
213
214 /* USER CODE BEGIN 4 */
215 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
216 {
217     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
218 }
219 /* USER CODE END 4 */
220
221 /**
222  * @brief This function is executed in case of error occurrence.
223  * @param file: The file name as string.
```

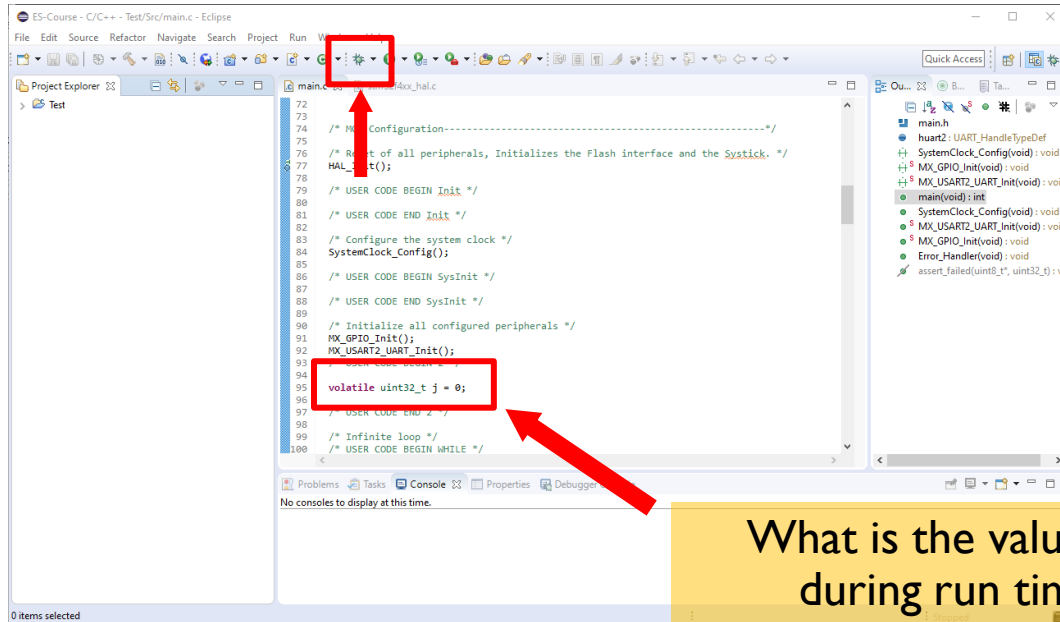
You can see that the troubles of before are solved. There aren't "false" reads and lags. Each Blue Button push generate 1 univocal Interrupt and a corresponding univocal Led toggle. (of course we can still have bouncing)



DEBUG

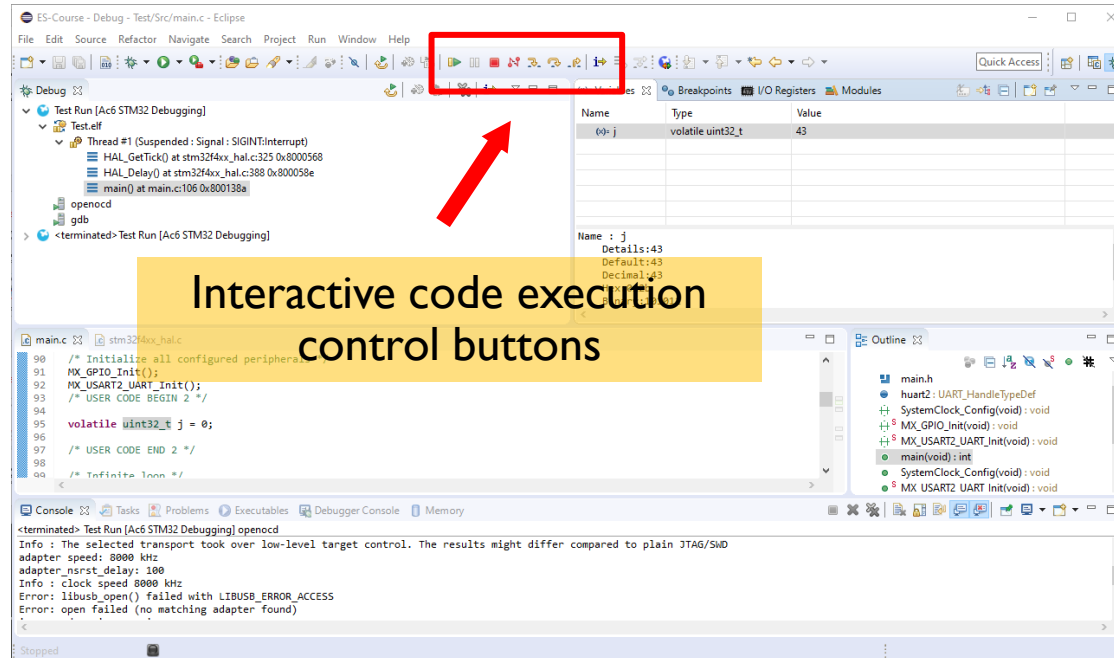
STM32 DEBUG

Sometimes we are interested in knowing what is happening inside our microcontroller.
We can start a debug session by pressing the bug icon



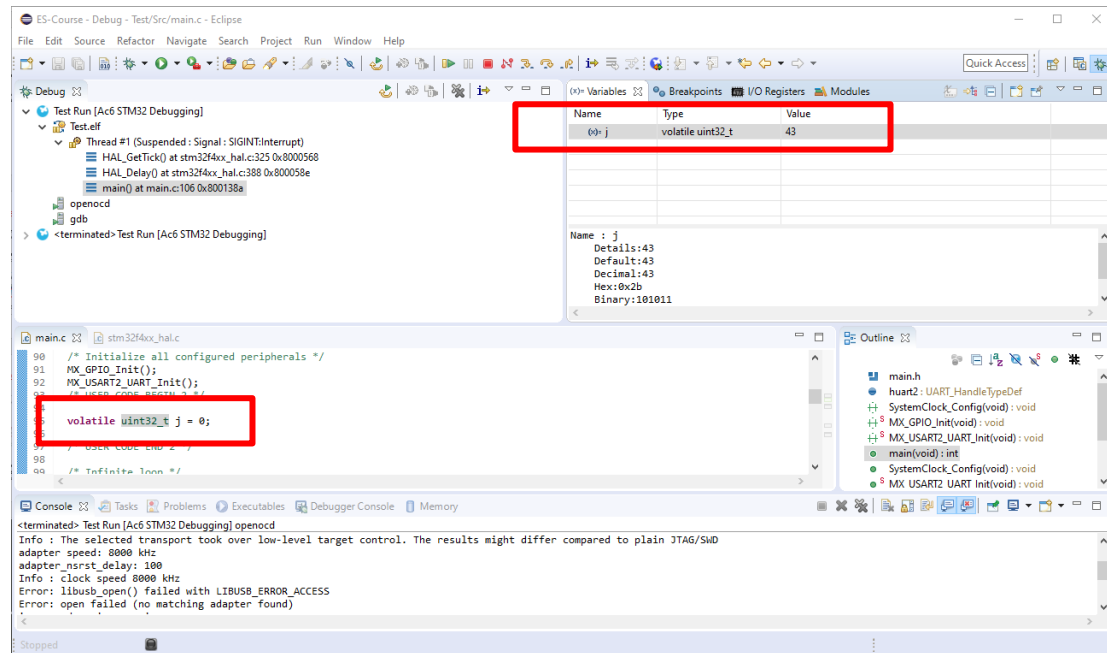
STM32 DEBUG

We can run the code step by step, or use breakpoints to stop the code at specific points



STM32 DEBUG

Debug functionality allow us to **interactively** run the code and to check register and variable actual values



STM32 DEBUG

We can also inspect I/O registers to check specific configuration or input/output values.

The screenshot shows the 'I/O Registers' window in a debugger. The 'I/O Registers' tab is selected and highlighted with a red box. The window displays a list of registers for the GPIOA peripheral. The 'ODR' (Output Data Register) is expanded, showing individual bits from ODR15 down to ODR0. A red box highlights the 'ODR5 [bit 5]' register, which has a hex value of 0x1 and a binary value of 1. A red arrow points to this bit. A yellow text box with the text 'When the LED is ON the associated register bit is set' is overlaid on the right side of the ODR register list.

Register	Hex value	Binary value	Reset value	Access	Address	Description
GPIOA					0x48000000	General-purpose I/Os
MODER	0xABFFFF7AF	10_10_10_11_11_11_11...	0xA8000000	READ-WRITE	0x48000000	GPIO port mode register
OTYPER			0x00000000	READ-WRITE	0x48000004	GPIO port output type register
OSPEEDR			0x00000000	READ-WRITE	0x48000008	GPIO port output speed register
PUPDR			0x64000000	READ-WRITE	0x4800000C	GPIO port pull-up/pull-down reg
IDR			0x00000000	READ-ONLY	0x48000010	GPIO port input data register
ODR	0x00000020	0000000000000000 0 0 0 ...	0x00000000	READ-WRITE	0x48000014	GPIO port output data register
ODR15 [bit 15]	0x0	0				Port output data (y = 0..15)
ODR14 [bit 14]	0x0	0				Port output data (y = 0..15)
ODR13 [bit 13]	0x0	0				Port output data (y = 0..15)
ODR12 [bit 12]	0x0	0				Port output data (y = 0..15)
ODR11 [bit 11]	0x0	0				Port output data (y = 0..15)
ODR10 [bit 10]	0x0	0				Port output data (y = 0..15)
ODR9 [bit 9]	0x0	0				Port output data (y = 0..15)
ODR8 [bit 8]	0x0	0				Port output data (y = 0..15)
ODR7 [bit 7]	0x0	0				Port output data (y = 0..15)
ODR6 [bit 6]	0x0	0				Port output data (y = 0..15)
ODR5 [bit 5]	0x1	1				Port output data (y = 0..15)
ODR4 [bit 4]	0x0	0				Port output data (y = 0..15)
ODR3 [bit 3]	0x0	0				Port output data (y = 0..15)
ODR2 [bit 2]	0x0	0				Port output data (y = 0..15)
ODR1 [bit 1]	0x0	0				Port output data (y = 0..15)
ODR0 [bit 0]	0x0	0				Port output data (y = 0..15)
BSRR			0x00000000	WRITE-ONLY	0x48000018	GPIO port bit set/reset register
LCKR			0x00000000	READ-WRITE	0x4800001C	GPIO port configuration lock reg