# LAB 02
# GPIO AND LED


UNIVERSITY OF TRENTO

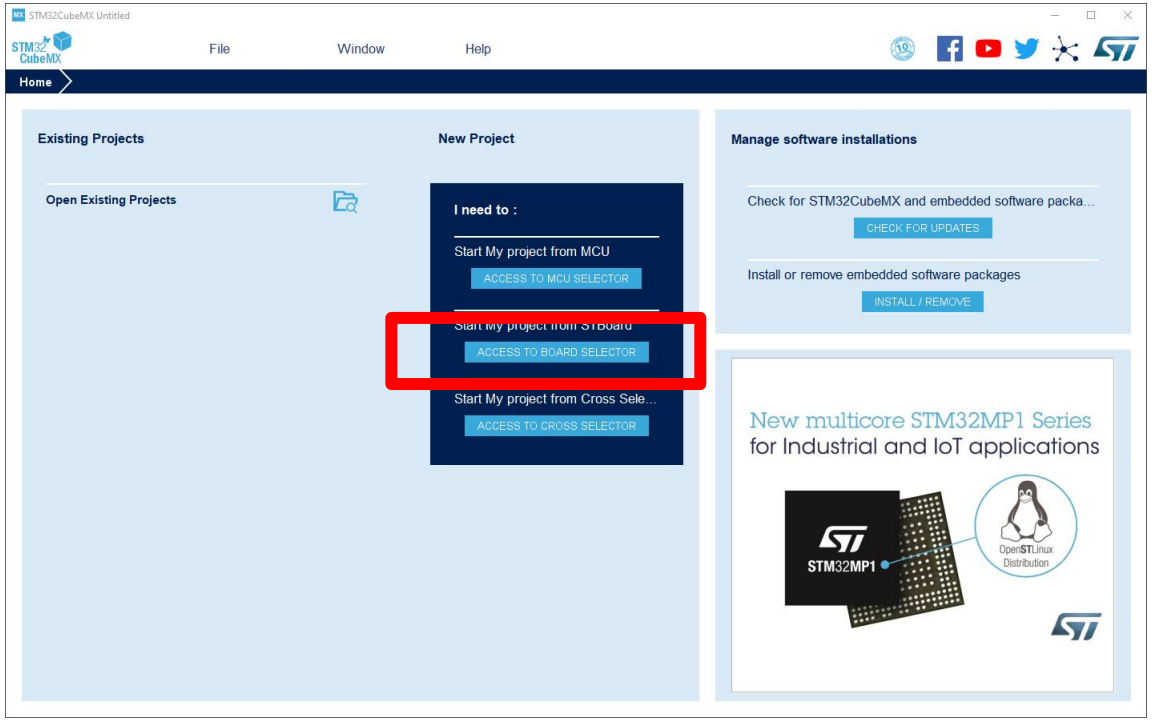
**Prof. Davide Brunelli**
Dept. of Industrial Engineering – DII
University of Trento, Italy
***davide.brunelli@unitn.it***

# STM32 CUBEMX

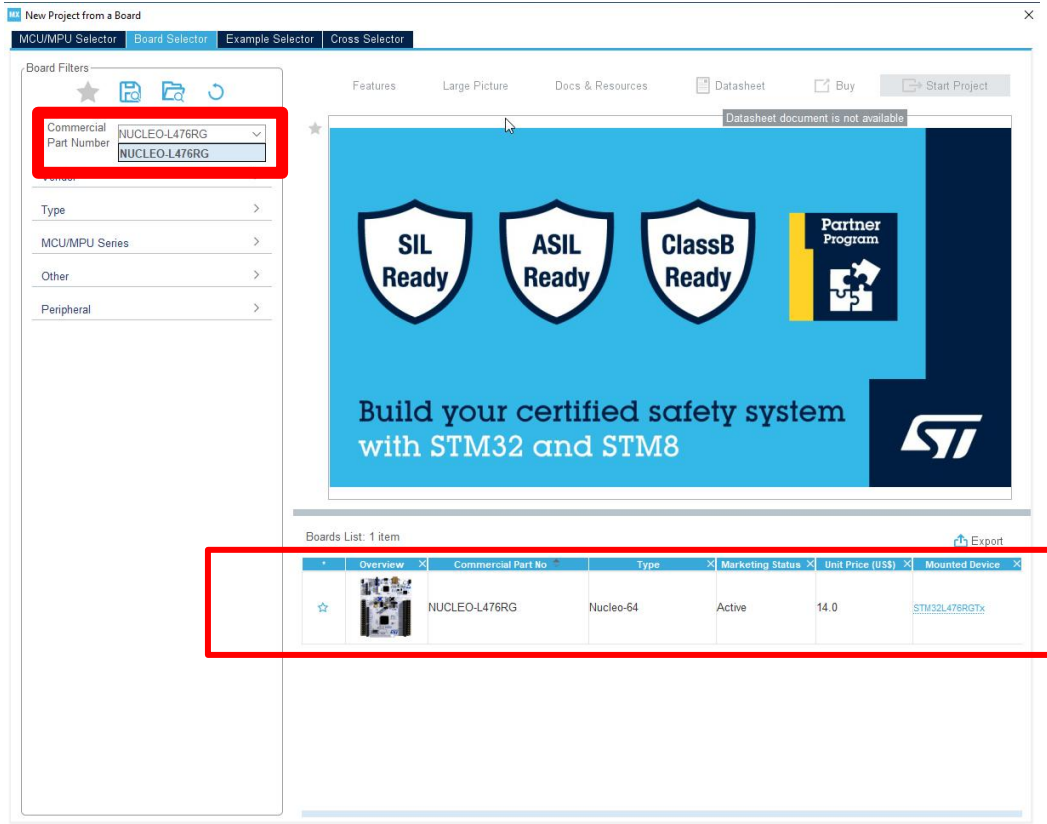
# STM32 CUBEMX – SELECTING THE BOARD

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

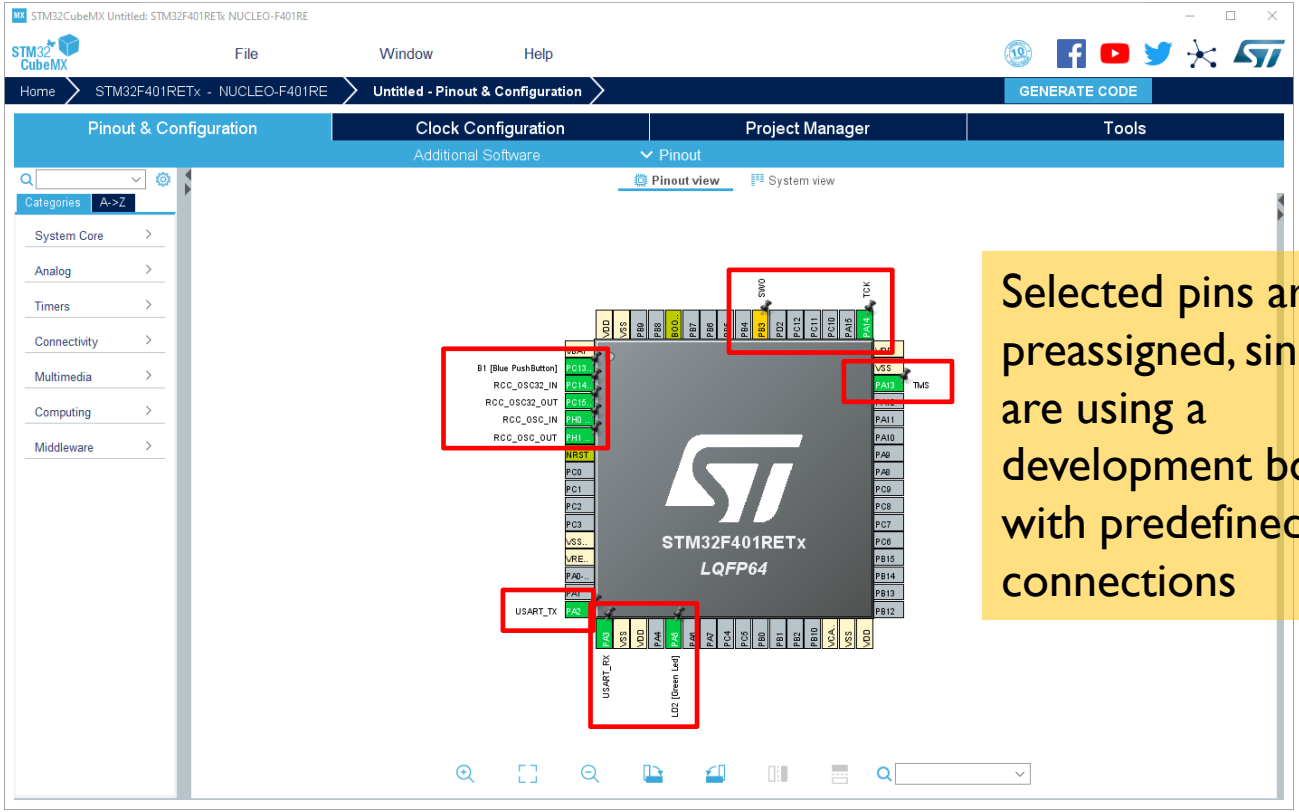
**Generates** the corresponding **C code** for the Arm Coretex core.



# STM32 CUBEMX – SELECTING THE BOARD

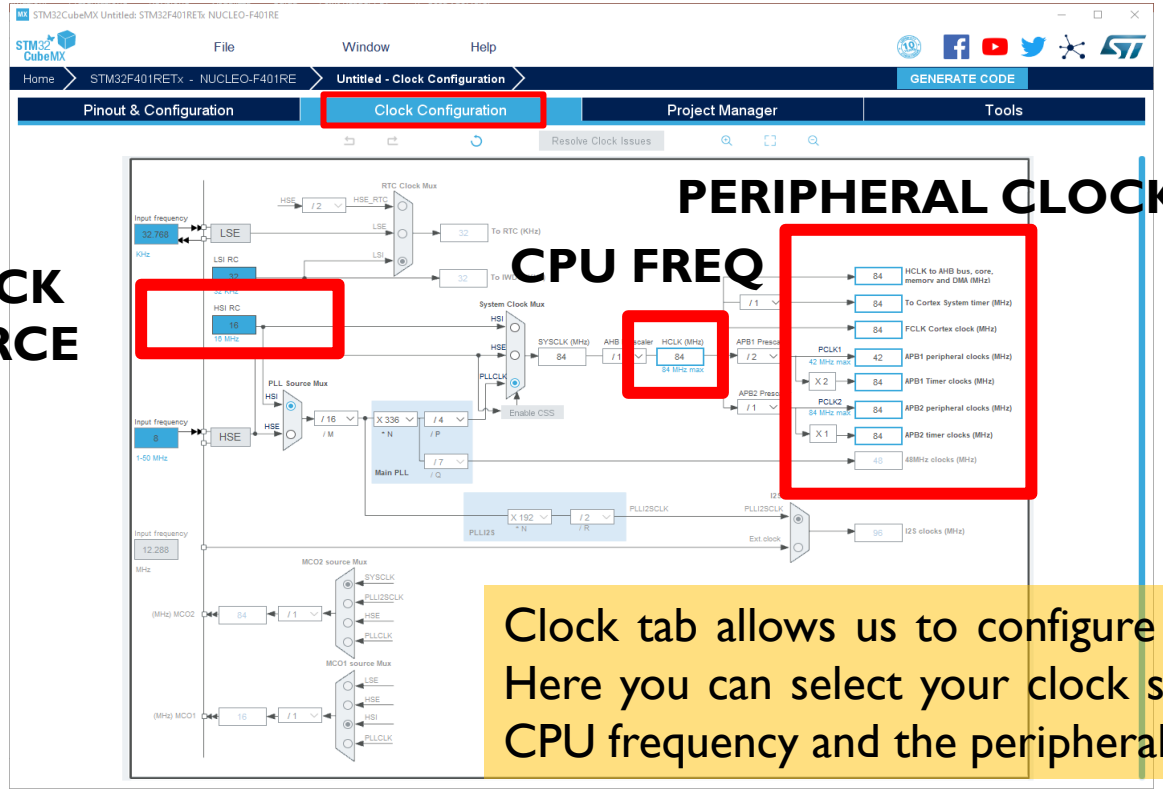


# STM32 CUBEMX – MULTIPLEXING THE PINS



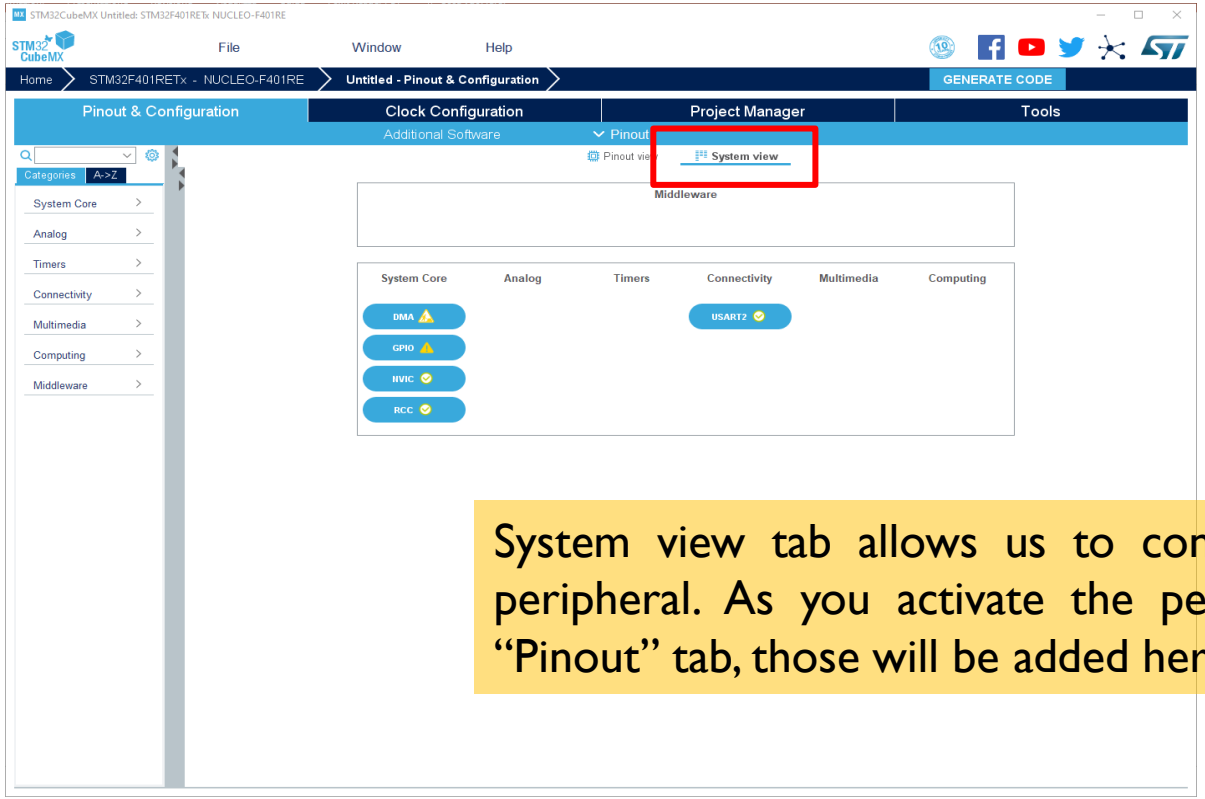

Selected pins are preassigned, since we are using a development board, with predefined connections

# STM32 CUBEMX – SETTING THE CLOCKS



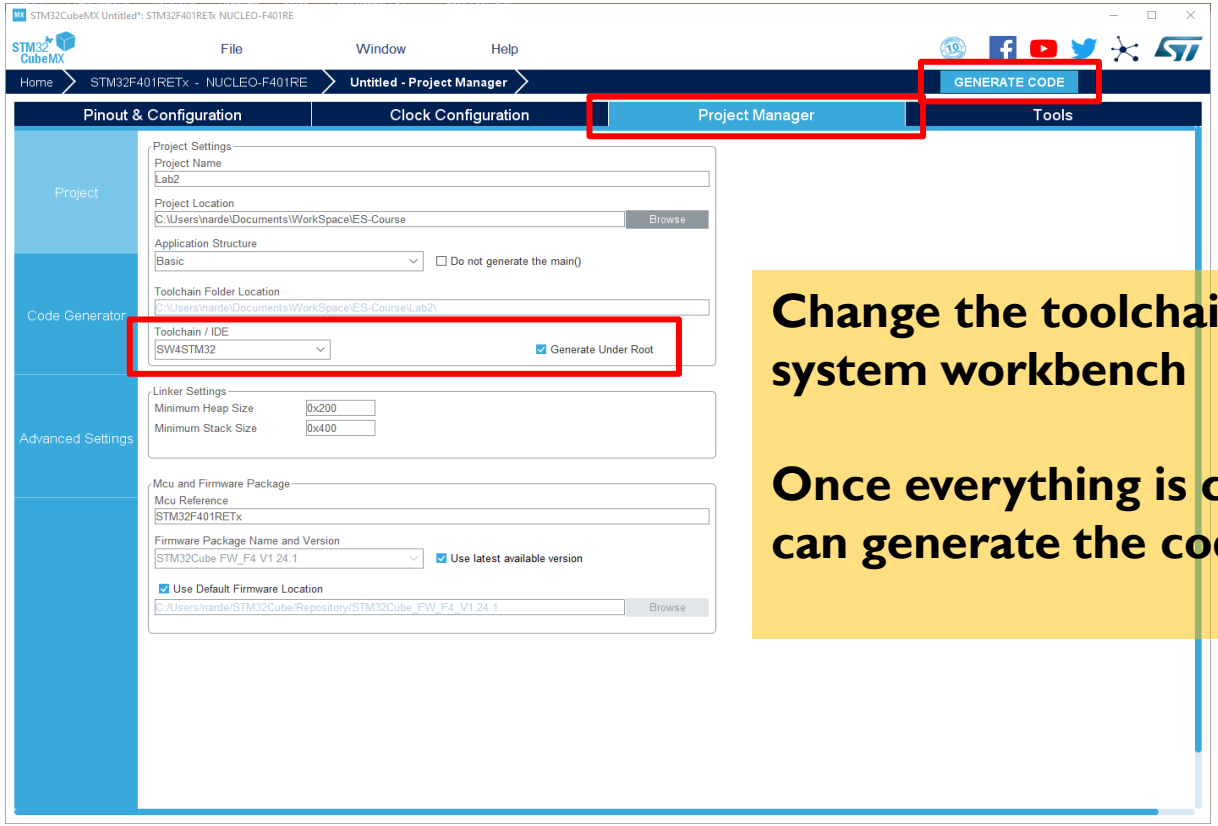

Clock tab allows us to configure the clock. Here you can select your clock source, the CPU frequency and the peripheral clock

# 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

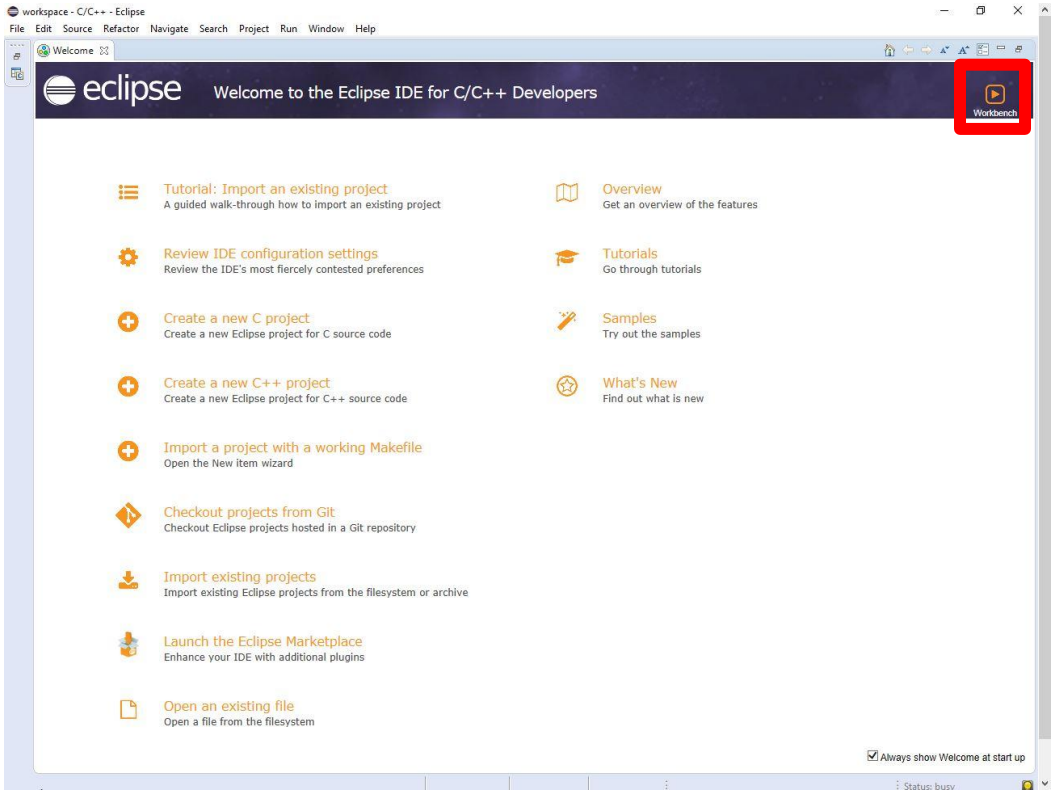



**Change the toolchain / IDE to system workbench**

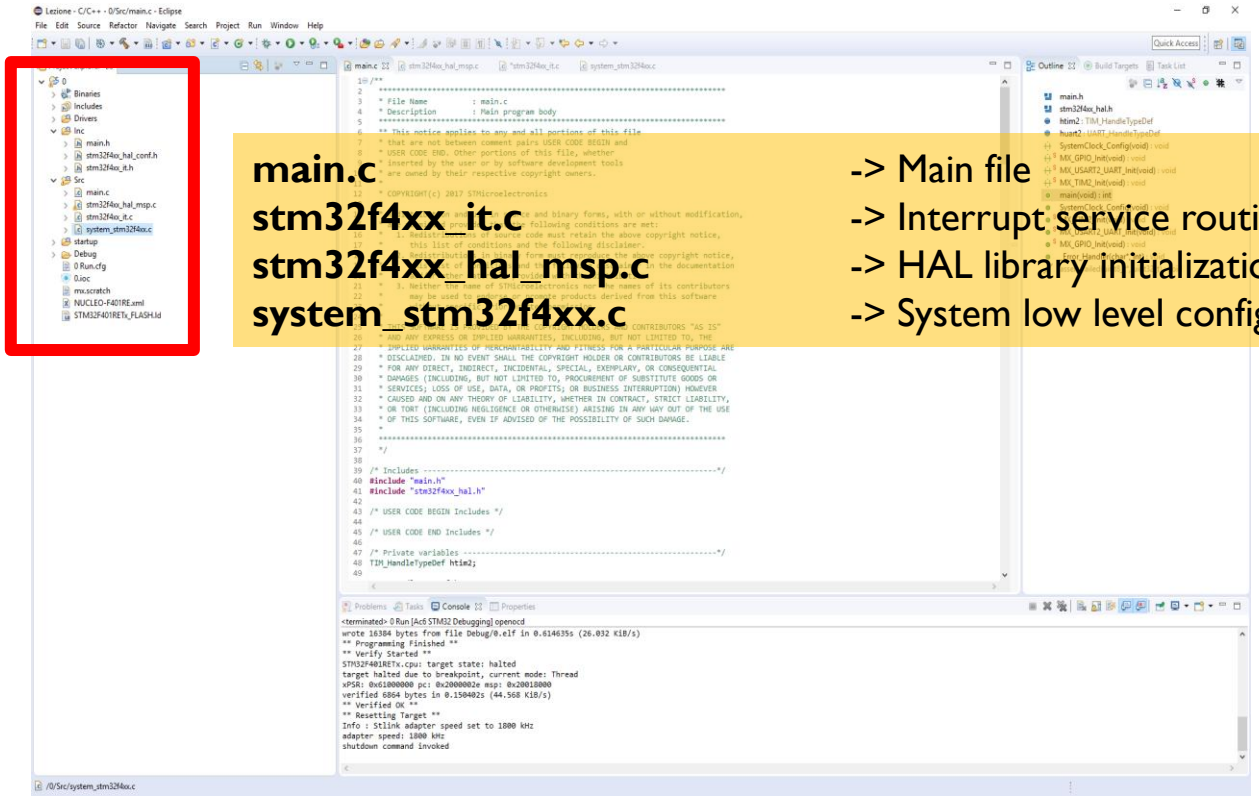
**Once everything is configure, we can generate the code**

# SYSTEMWORKBENCH FOR STM32

# STM32 – SYSTEM WORKBENCH

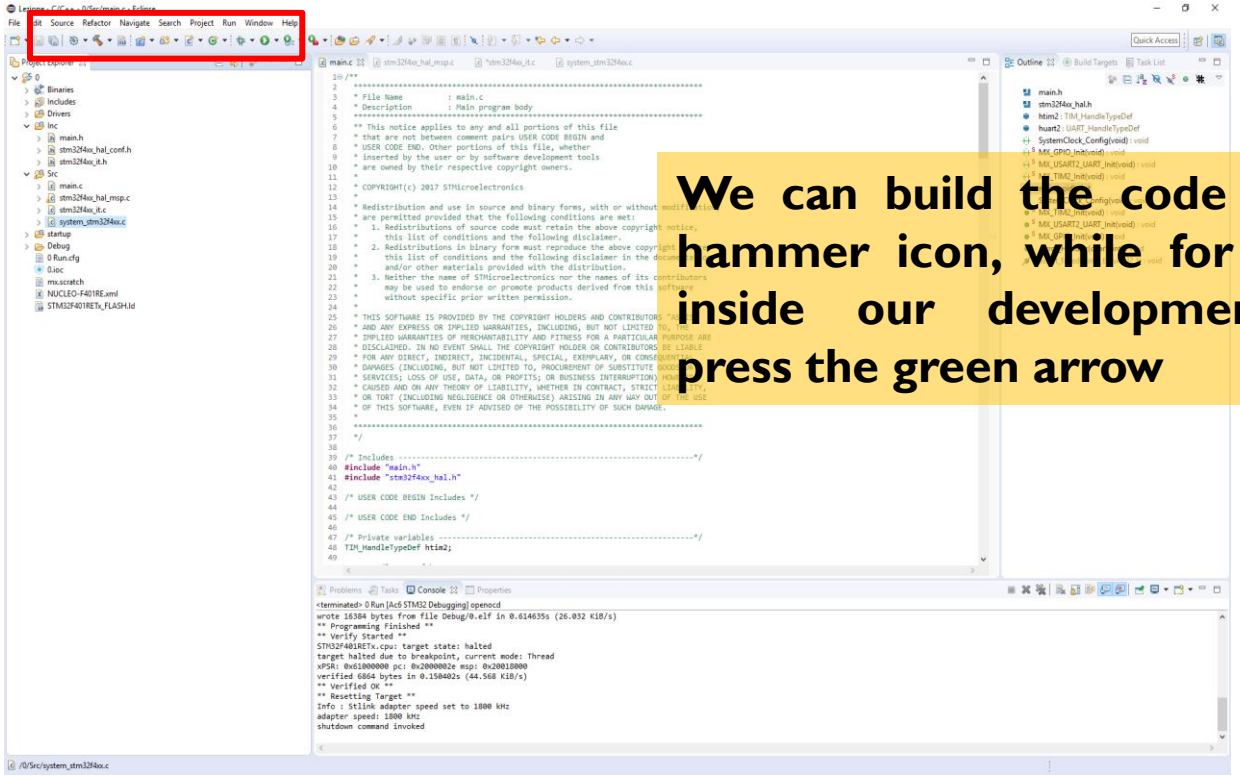


# STM32 – SYSTEM WORKBENCH



**main.c** -> Main file
**stm32f4xx_it.c** -> Interrupt service routine
**stm32f4xx_hal_msp.c** -> HAL library initialization
**system_stm32f4xx.c** -> System low level config

# STM32 – SYSTEM WORKBENCH




**We can build the code pressing the hammer icon, while for executing it inside our development platform, press the green arrow**

# 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 1 – 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 (B1)** 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 se how to switch the led in 3 way:**

- **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

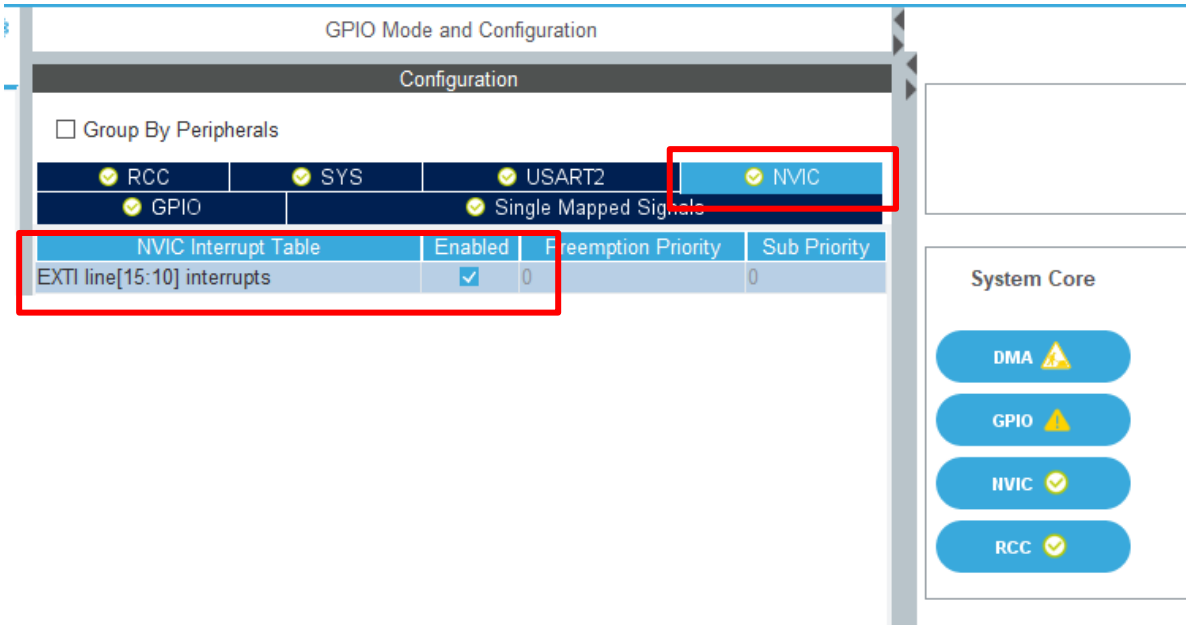
Checking the LED, you can notice that the behavior isn't always as expected: most times it toggles correctly, but sometimes the LED looks like switching ON (or OFF), erroneously. This because we can miss Blue Button press event or we can double read it.

To solve this trouble, we introduce the **Interrupt** technique. The generates a signal alerting the system of the Pin 13 activation. When the Interrupts is received, the Led toggles.

# STM32 GPIO – TOGGLE BY INTERRUPT

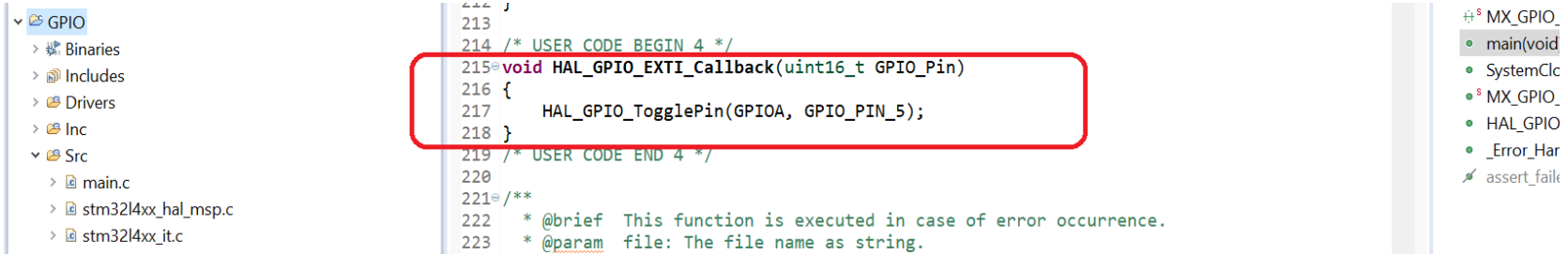
To enable the interrupt, go back to cubeMX and under **NVIC** activate Interrupts **EXTI Line [15:10]**.

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

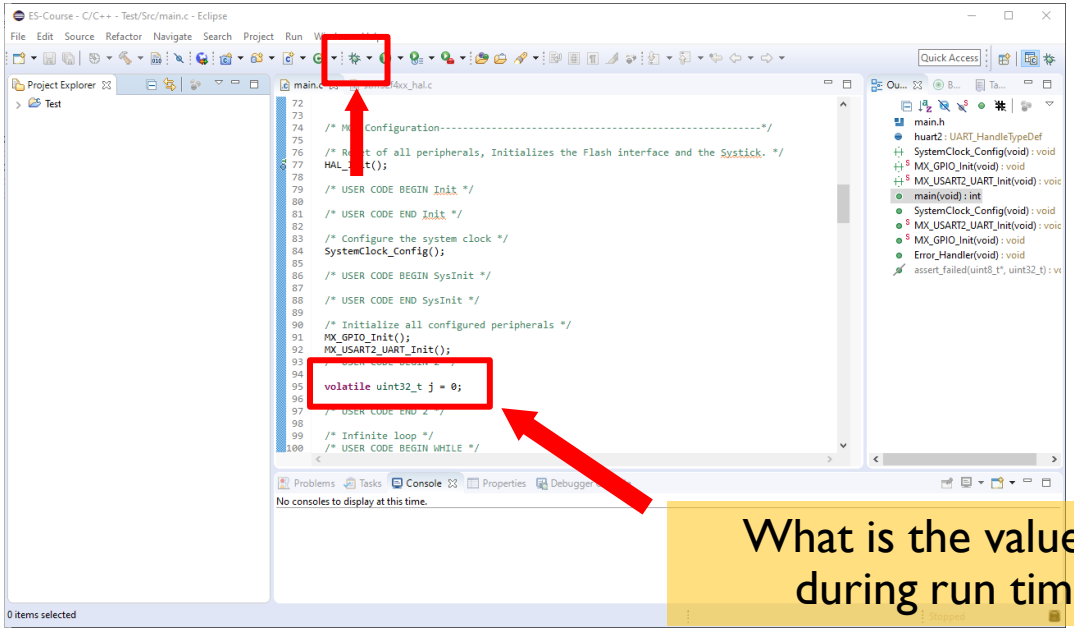



You can see that the troubles of before are solved. There aren't "false" reads and lags. Each Blue Button push generate 1 univocal Interrupt and a corresponding univocal Led toggle. (of course we can still have bouncing)

# DEBUG

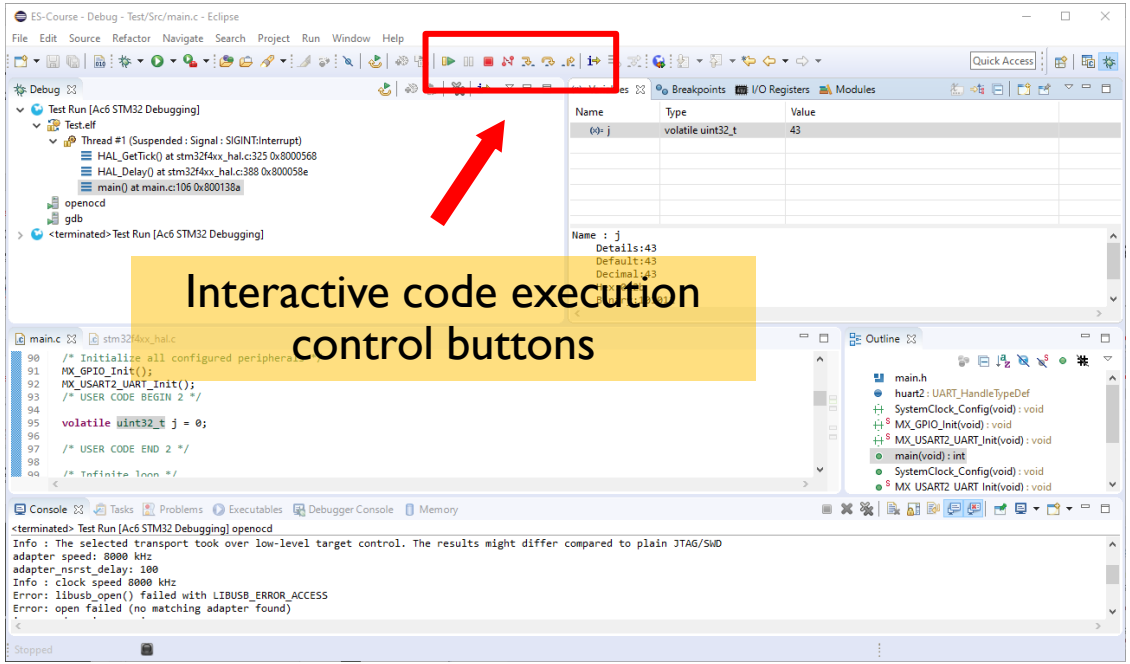
# STM32 DEBUG

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



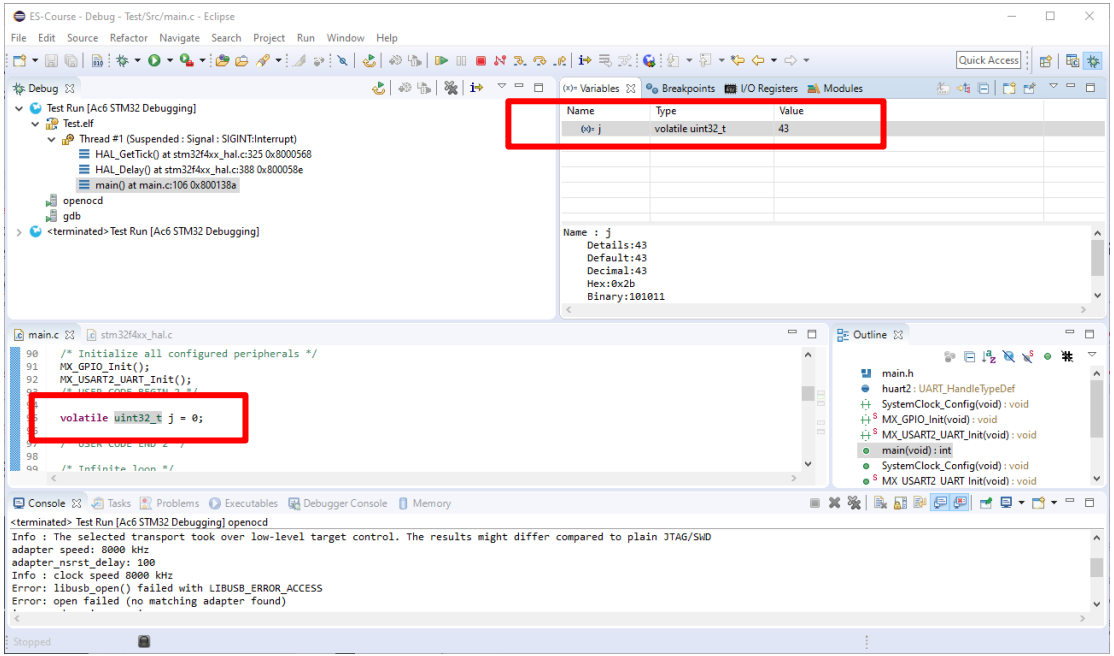
# STM32 DEBUG

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



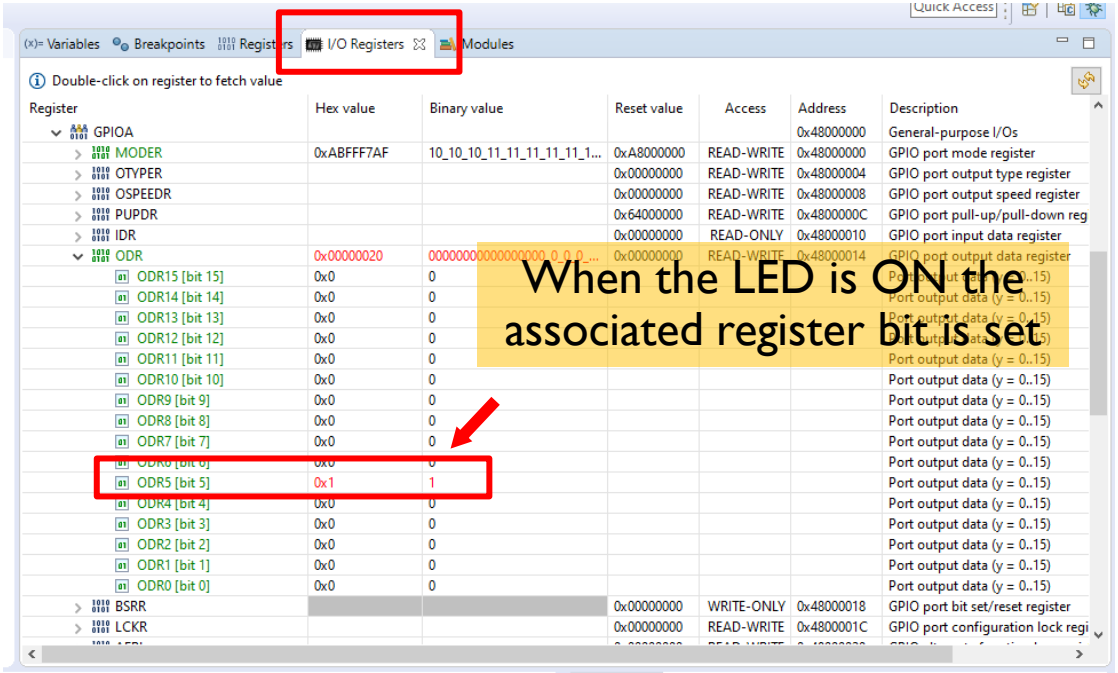
# STM32 DEBUG

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.




| Register | Hex value | Binary value | Reset value | Access | Address | Description |
|---|---|---|---|---|---|---|
| GPIOA | | | | | 0x48000000 | General-purpose I/Os |
| MODER | 0xABFFF7AF | 10_10_10_11_11_11_11_11_1... | 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 regi |