Blink Sketch con LL libraries

rev1.0 24/03/2020

GOAL

Toggle a GPIO Output using LL Libraries

PREREQUISITES

Software needed:

- STM32IDE

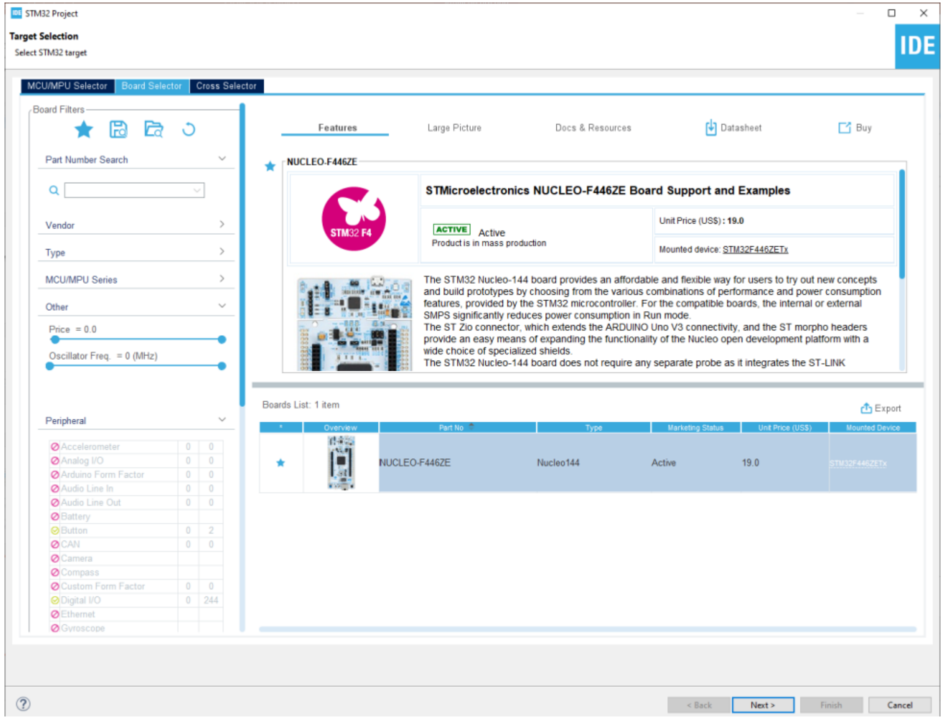
Hardware used in this example:

- **NUCLEO-F446ZE**

Start a new project

From the stm32IDE software click on File -> New -> STM32 Project.

Select your board or your uC and click *next.*



NUCLEO-F446ZE

STMicroelectronics NUCLEO-F446ZE Board Support and Examples

Active
Product is in mass production

Unit Price (US$) : **19.0**

Mounted device: STM32F446ZETx

The STM32 Nucleo-144 board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features, provided by the STM32 microcontroller. For the compatible boards, the internal or external SMPS significantly reduces power consumption in Run mode.
The ST Zio connector, which extends the ARDUINO Uno V3 connectivity, and the ST morpho headers provide an easy means of expanding the functionality of the Nucleo open development platform with a wide choice of specialized shields.
The STM32 Nucleo-144 board does not require any separate probe as it integrates the ST-LINK

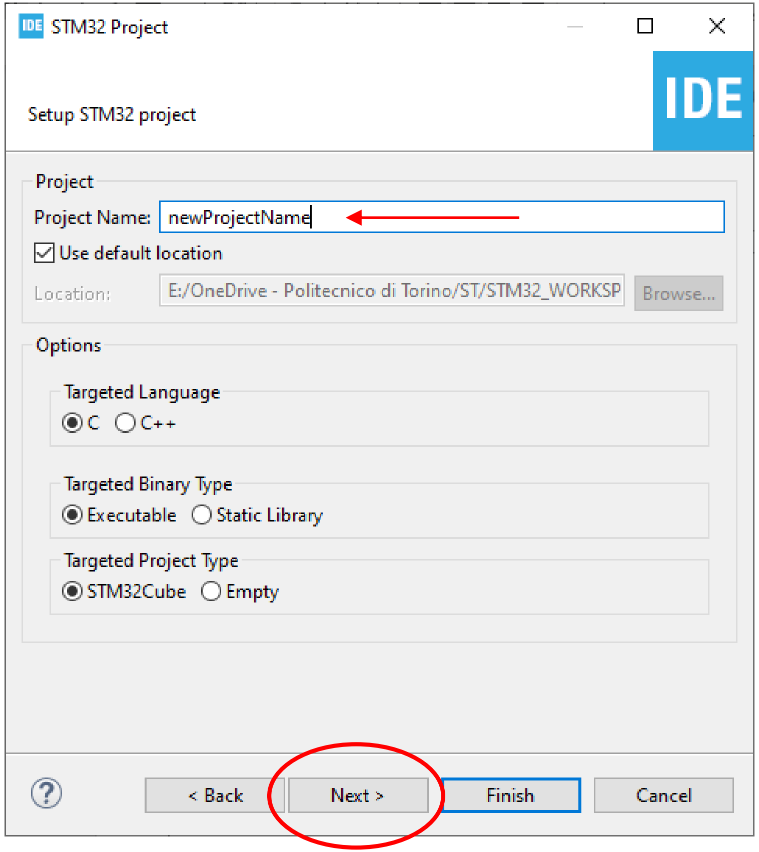
Boards List: 1 item

*	Overview	Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
★		NUCLEO-F446ZE	Nucleo144	Active	19.0	STM32F446ZETx

Start a new project

Type the name of your project and click next.

By default the project will be created in the workspace folder.



STM32 Project

Setup STM32 project

Project

Project Name: newProjectName

[✓] Use default location

Location: E:/OneDrive - Politecnico di Torino/ST/STM32_WORKSP

Options

Targeted Language: [● C] ○ C++

Targeted Binary Type: [● Executable] ○ Static Library

Targeted Project Type: [● STM32Cube] ○ Empty

Start a new project

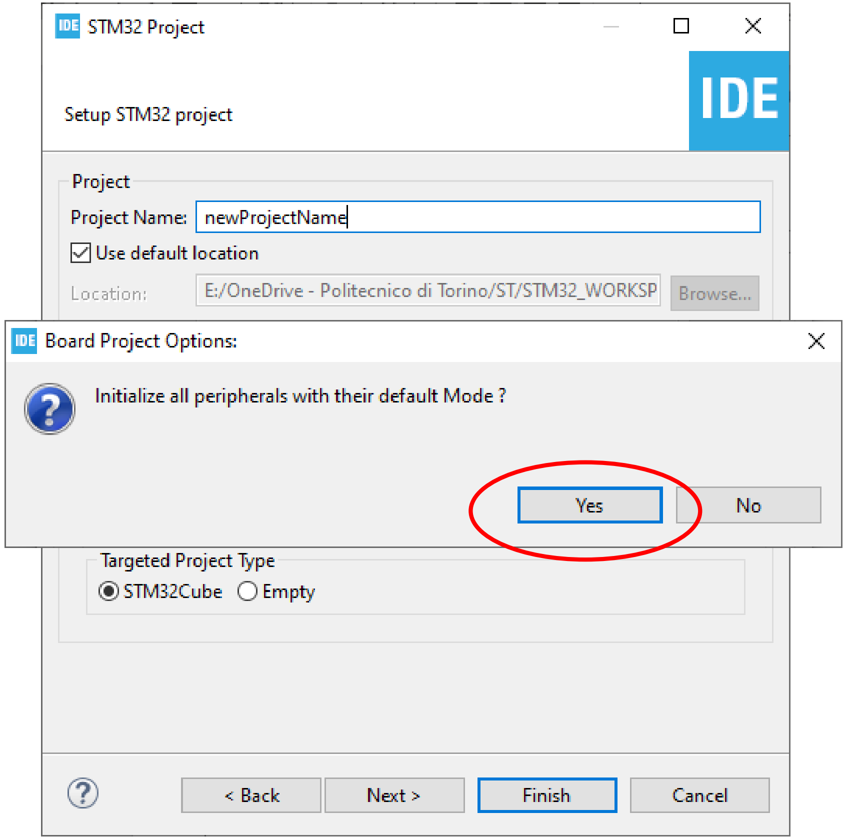
Type the name of your project and click next.

By default the project will be created in the *workspace* folder.

The *STM32IDE* has the option to initialize all the peripheral with their ***default*** mode:

Clicking *Yes* the *USART3*, all the *LEDs* and the blue *UserButton* will be configured as default.

Click *Yes*.


STM32 Project

Setup STM32 project

Project

Project Name: newProjectName

☑ Use default location

Location: E:/OneDrive - Politecnico di Torino/ST/STM32_WORKSP

Board Project Options:

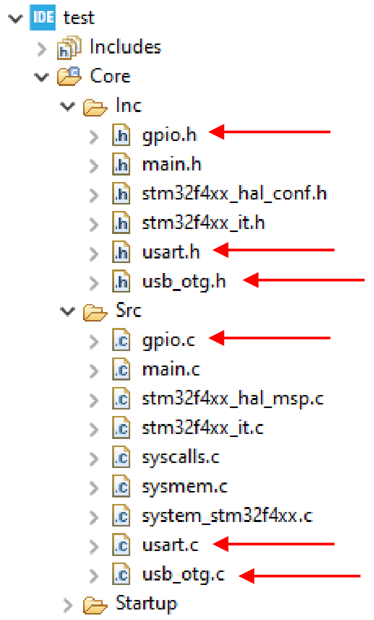
Initialize all peripherals with their default Mode ?

Targeted Project Type

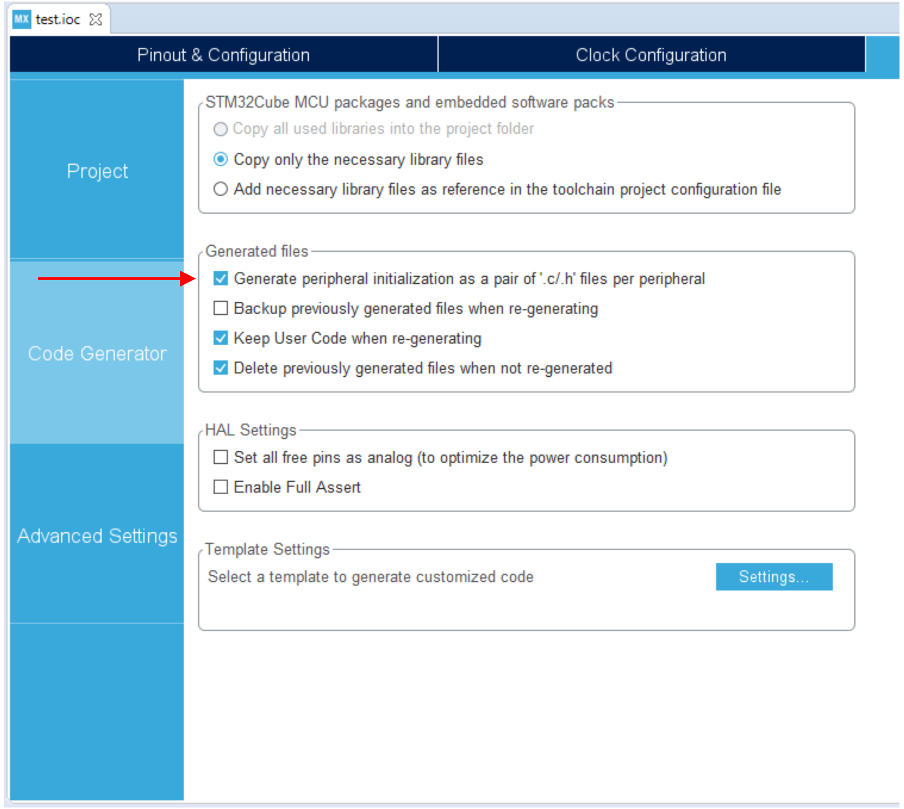
● STM32Cube ○ Empty

Project Manager

In the Code Generator Tab check the ***Generate peripheral initialization […]*** box: each periperhal will have a disting *periph.c* and *periph.h* files.



```
test
  Includes
  Core
    Inc
      gpio.h
      main.h
      stm32f4xx_hal_conf.h
      stm32f4xx_it.h
      usart.h
      usb_otg.h
    Src
      gpio.c
      main.c
      stm32f4xx_hal_msp.c
      stm32f4xx_it.c
      syscalls.c
      sysmem.c
      system_stm32f4xx.c
      usart.c
      usb_otg.c
    Startup
```



STM32Cube MCU packages and embedded software packs
- ○ Copy all used libraries into the project folder
- ● Copy only the necessary library files
- ○ Add necessary library files as reference in the toolchain project configuration file

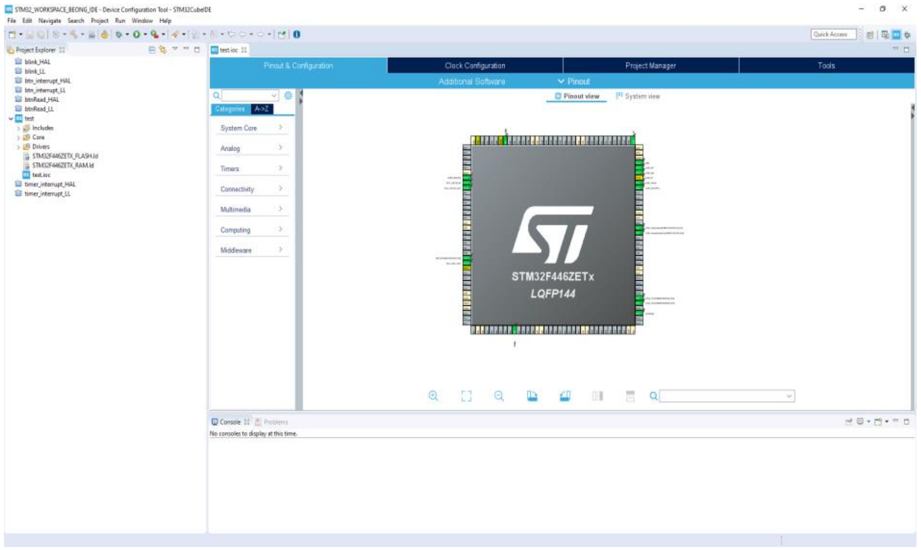
Generated files
- ☑ Generate peripheral initialization as a pair of '.c/.h' files per peripheral
- ☐ Backup previously generated files when re-generating
- ☑ Keep User Code when re-generating
- ☑ Delete previously generated files when not re-generated

HAL Settings
- ☐ Set all free pins as analog (to optimize the power consumption)
- ☐ Enable Full Assert

Template Settings
Select a template to generate customized code

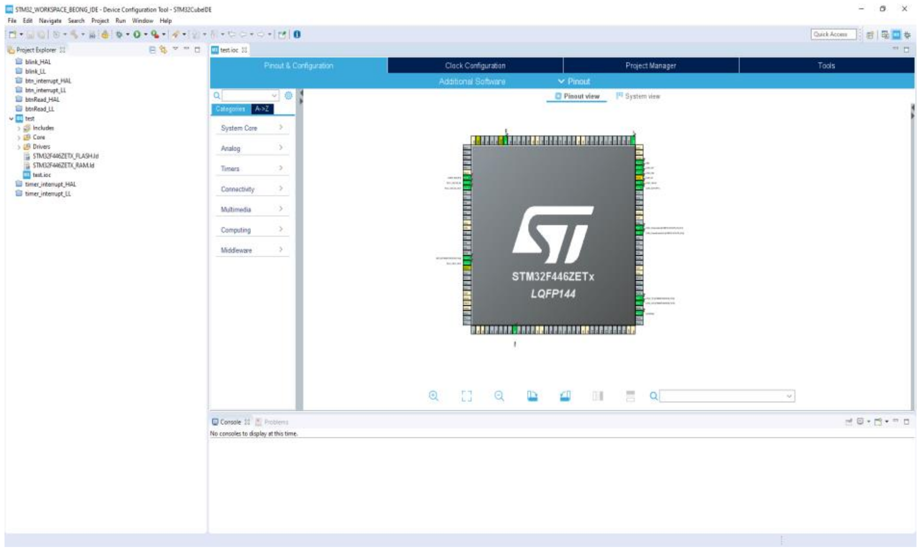
GPIO: General Pourpose Input/Output

- Dopo aver creato il progetto e inizializzato le periferiche ai propri valori di default, è utile, prima di iniziare a programmare, scorrere velocemente le caratteristiche dei pin GPIO e come possono essere configurati.
- Cliccando su un pin generico con il tasto sinistro del mouse apparirà una lista con tutte le possibili configurazione dei pin.



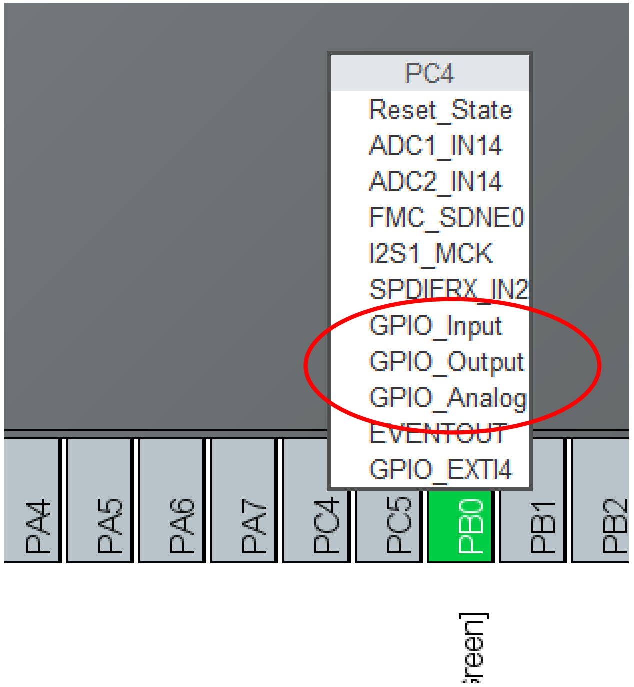
GPIO: General Pourpose Input/Output

- After creating the project and initializing the functions to their default values, it is useful, before starting to scroll, to quickly scroll through the characteristics of the GPIO pins and how to be configured.
- By clicking on a generic pin with the left mouse button a list will appear with all the pin configuration options.

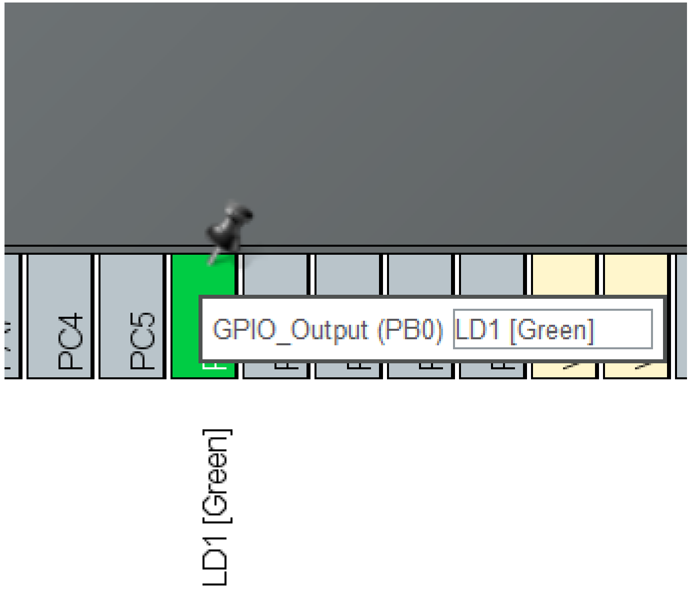


No consoles to display at this time.

- In this example we will analyze in particular the ***GPIO_xx***:
- *GPIO_OUTPUT*: Used to control an output device (e.g. LED, Motor, etc.)
- *GPIO_INPUT*: It is used in case you want to read the state of an input device (eg Switch, Sensors, etc.)
- *GPIO_ANALOG*: Similar to the input, it is used when the input device is analog (e.g. Photoresistor, Potentiometer, etc.)

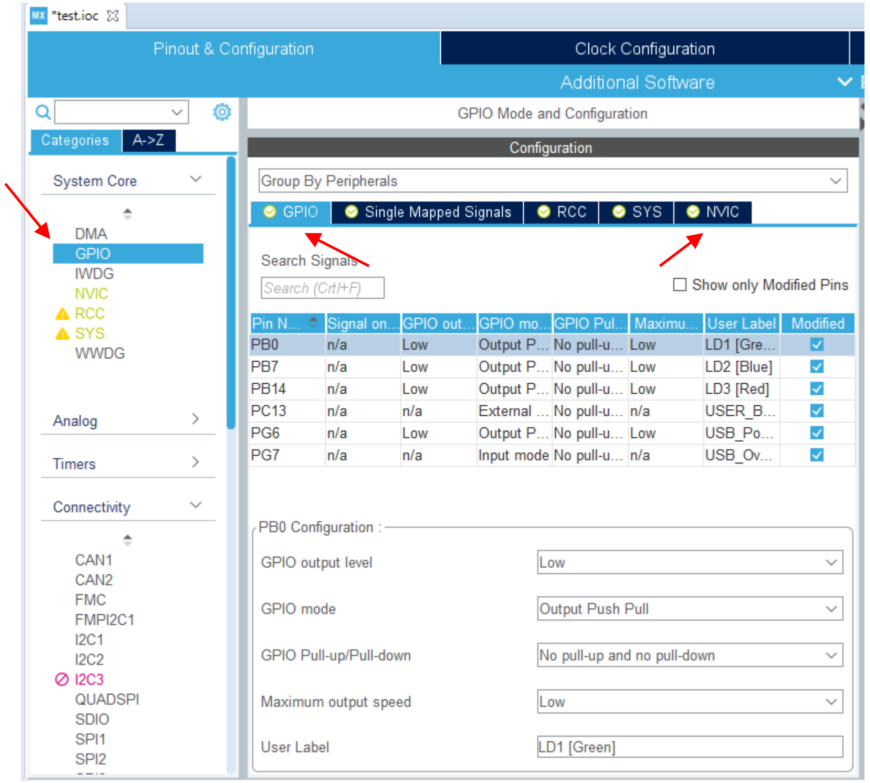


- By right-clicking on an already set pin it is instead possible to insert a **label**, very useful when writing the code, when you have numerous peripherals to manage.
- Under the heading ***system core → GPIO*** we find a more in-depth view of GPIOs



GPIO_Output (PB0) LD1 [Green]

- Below the ***GPIO*** tab we find the list of the set pins and their characteristics specifically.
- Further down we find the *Configuration* section where you can set other parameters such as default output level, mode, etc.
- Important, among the upper Tabs, we find the ***NVIC*** tab, useful for setting the interrupts which we will discuss later.

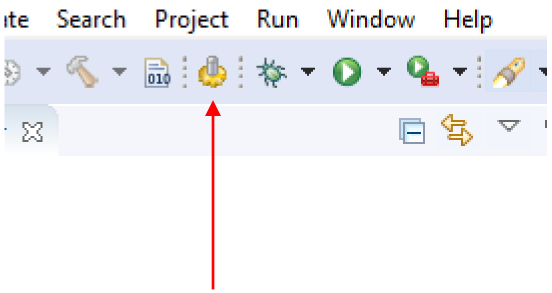


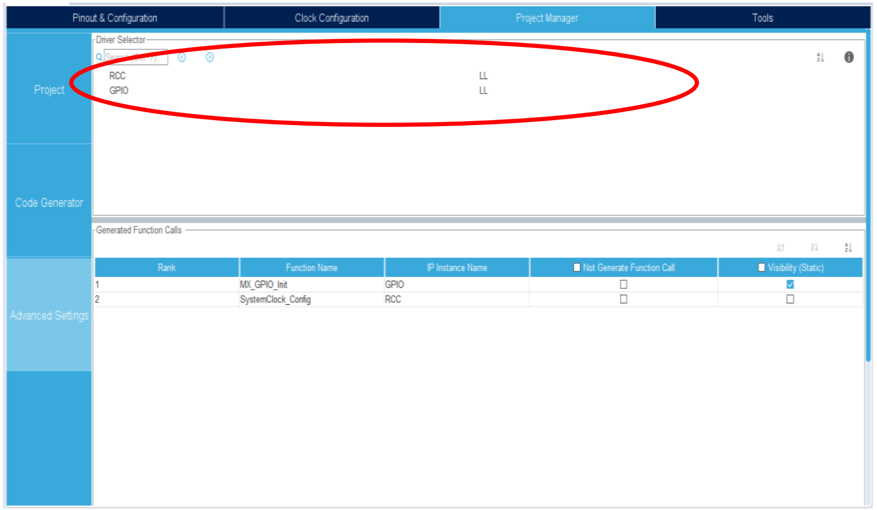
Pin N...	Signal on...	GPIO out...	GPIO mo...	GPIO Pul...	Maximu...	User Label	Modified
PB0	n/a	Low	Output P...	No pull-u...	Low	LD1 [Gre...	✓
PB7	n/a	Low	Output P...	No pull-u...	Low	LD2 [Blue]	✓
PB14	n/a	Low	Output P...	No pull-u...	Low	LD3 [Red]	✓
PC13	n/a	n/a	External ...	No pull-u...	n/a	USER_B...	✓
PG6	n/a	Low	Output P...	No pull-u...	Low	USB_Po...	✓
PG7	n/a	n/a	Input mode	No pull-u...	n/a	USB_Ov...	✓

PB0 Configuration :

GPIO output level	Low
GPIO mode	Output Push Pull
GPIO Pull-up/Pull-down	No pull-up and no pull-down
Maximum output speed	Low
User Label	LD1 [Green]

- At this point, all that remains is to verify that you have chosen the **HAL** libraries for managing the GPIO and continue with the generation of the code using the appropriate icon.





Driver Selector

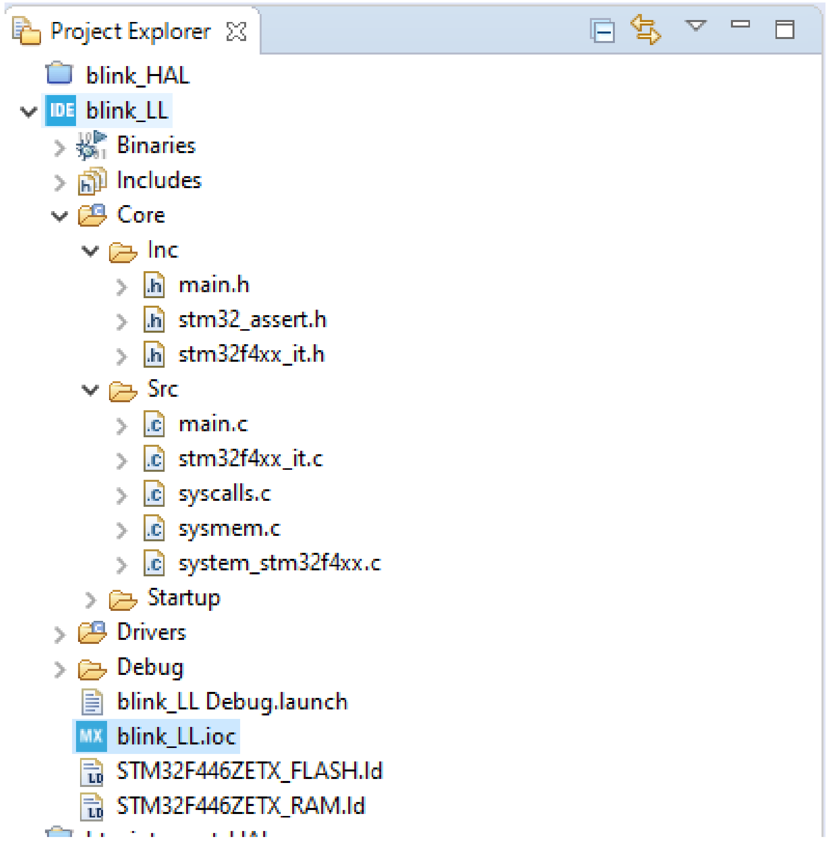
RCC	LL
GPIO	LL

Generated Function Calls

Rank	Function Name	IP Instance Name	☐ Not Generate Function Call	☐ Visibility (Static)
1	MX_GPIO_Init	GPIO	☐	☑
2	SystemClock_Config	RCC	☐	☐

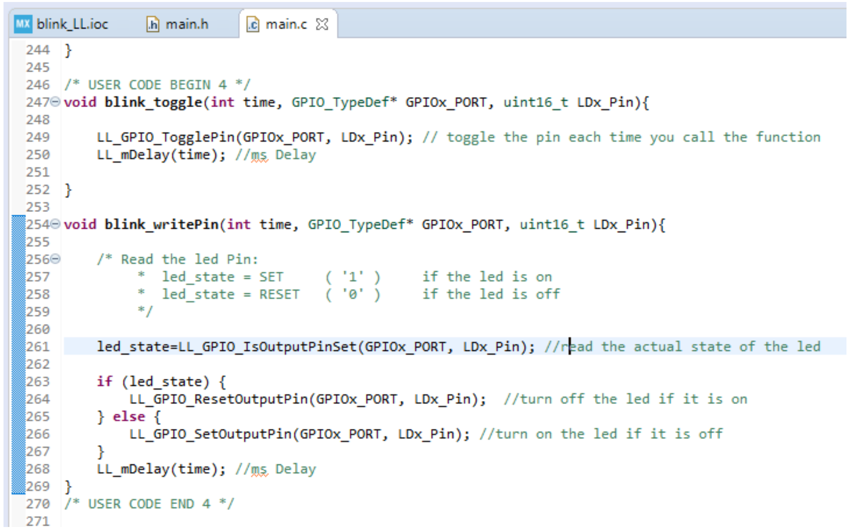
Iniziamo a Programmare

- Una volta generato il codice, sulla sinistra troviamo il ***project explorer***: da qui abbiamo una panoramica su tutti i file che compongono il nostro progetto.
- Iniziamo con l'aprire il ***main.h*** e il ***main.c***



User code BEGIN and END

- Inside files generated by **CubeMX** we will find spaces where the user can write some code: these spaces are marked with comments ***/ * USER CODE BEGIN X * / and / * USER CODE END X * /***
- These will be the only fields that will not be touched after a possible regeneration of the code by CubeMX, therefore it is **essential** not to write outside these spaces.
- In **main.c**, in section 4, you can add the functions written by the user, let's see them in more detail.

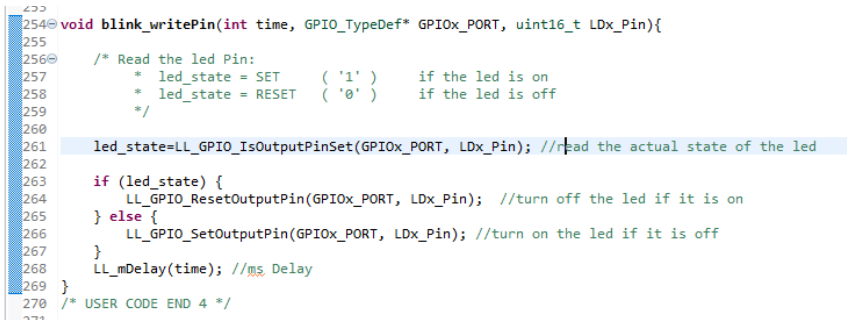


```c
}

/* USER CODE BEGIN 4 */
void blink_toggle(int time, GPIO_TypeDef* GPIOx_PORT, uint16_t LDx_Pin){

    LL_GPIO_TogglePin(GPIOx_PORT, LDx_Pin); // toggle the pin each time you call the function
    LL_mDelay(time); //ms Delay

}

void blink_writePin(int time, GPIO_TypeDef* GPIOx_PORT, uint16_t LDx_Pin){

    /* Read the led Pin:
         *  led_state = SET     ( '1' )     if the led is on
         *  led_state = RESET   ( '0' )     if the led is off
         */

    led_state=LL_GPIO_IsOutputPinSet(GPIOx_PORT, LDx_Pin); //read the actual state of the led

    if (led_state) {
        LL_GPIO_ResetOutputPin(GPIOx_PORT, LDx_Pin);  //turn off the led if it is on
    } else {
        LL_GPIO_SetOutputPin(GPIOx_PORT, LDx_Pin); //turn on the led if it is off
    }
    LL_mDelay(time); //ms Delay
}
/* USER CODE END 4 */

```

blink_writePin()

To make the led on the board flash, you can proceed in different ways:

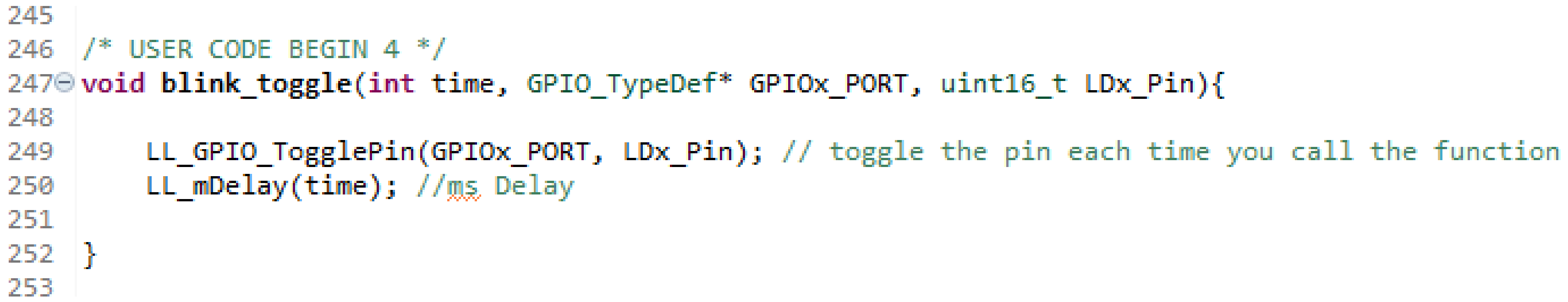
- A first strategy is to read the current state of the pin using the **LL_GPIO_IsOutputPinSet** function and consequently use the **LL_GPIO_ResetOutputPin ()** function to reset the output, use the function **LL_GPIO_SetOutputPin()** to set *output* instead :
  - *SET* -> ‘1’
  - *RESET* -> ‘0’



```c
void blink_writePin(int time, GPIO_TypeDef* GPIOx_PORT, uint16_t LDx_Pin){

    /* Read the led Pin:
        *  led_state = SET     ( '1' )     if the led is on
        *  led_state = RESET   ( '0' )     if the led is off
        */

    led_state=LL_GPIO_IsOutputPinSet(GPIOx_PORT, LDx_Pin); //read the actual state of the led

    if (led_state) {
        LL_GPIO_ResetOutputPin(GPIOx_PORT, LDx_Pin);  //turn off the led if it is on
    } else {
        LL_GPIO_SetOutputPin(GPIOx_PORT, LDx_Pin); //turn on the led if it is off
    }
    LL_mDelay(time); //ms Delay
}
/* USER CODE END 4 */
```

blink_toggle()

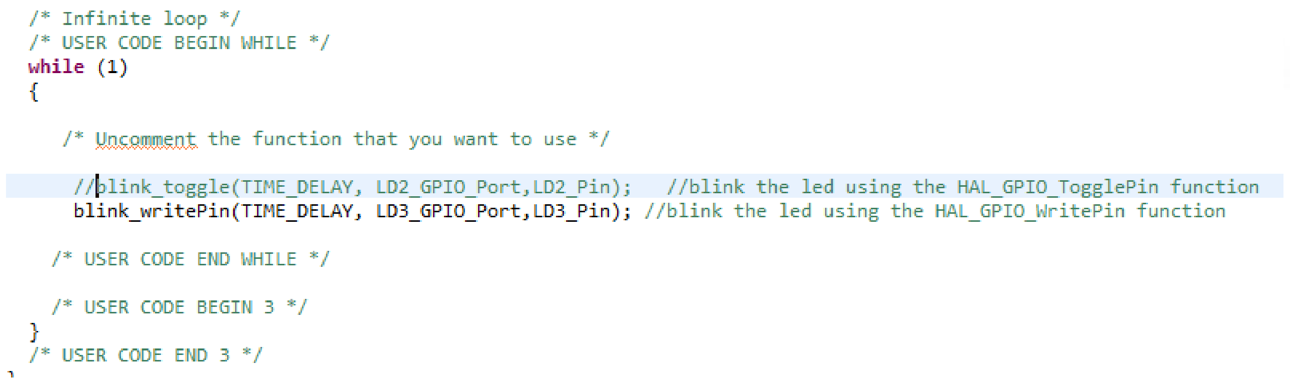
- Another way to change the output of an output pin may be to use the **LL_GPIO_TogglePin ()** function**;**
- This does nothing but read the current state of the pin (LDx_pin) and invert its value:
    - If *SET* -> *RESET*
    - If *RESET* -> *SET*



```
/* USER CODE BEGIN 4 */
void blink_toggle(int time, GPIO_TypeDef* GPIOx_PORT, uint16_t LDx_Pin){

    LL_GPIO_TogglePin(GPIOx_PORT, LDx_Pin); // toggle the pin each time you call the function
    LL_mDelay(time); //ms Delay

}
```

Main loop!

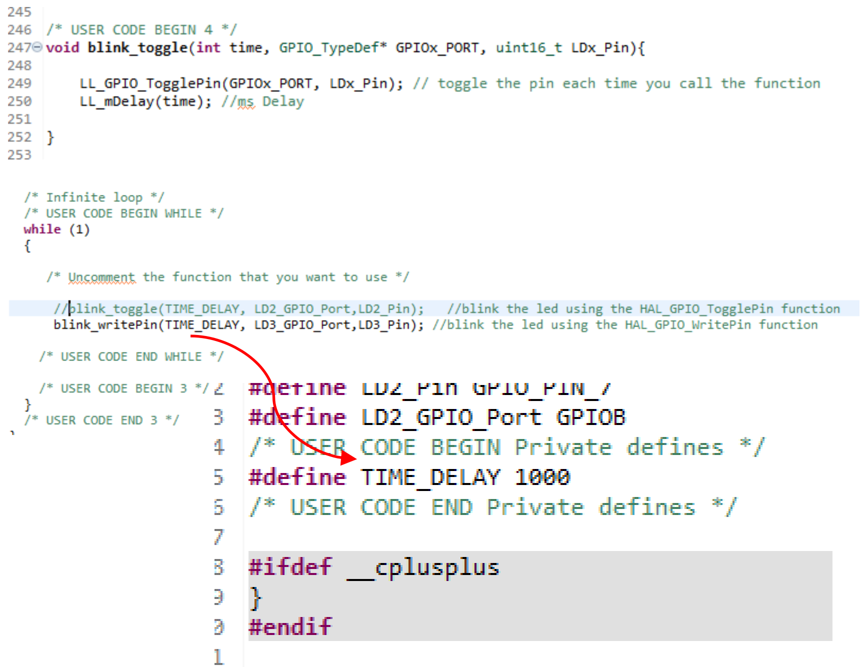
- Within the *main* function, after the various initializations that CubeMX has done for us, we find the *mainLoop*: the program will do nothing but repeat what we will write within it indefinitely.
- At this point, to make the LED flash, it is not enough to recall the functions we have just written!



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

     /* Uncomment the function that you want to use */

      //blink_toggle(TIME_DELAY, LD2_GPIO_Port,LD2_Pin);    //blink the led using the HAL_GPIO_TogglePin function
      blink_writePin(TIME_DELAY, LD3_GPIO_Port,LD3_Pin); //blink the led using the HAL_GPIO_WritePin function

    /* USER CODE END WHILE */

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

Tips & Tricks

- It is good practice to write functions that are as versatile as possible, for this we use **parameters**.
- It is also useful to use main.h to write the prototypes of the functions and also to define constants, such as the one we called TIME_DELAY so that we can use the same parameter several times in the code.
- It is possible to use the ***CRTL + SPACE*** shortcut for the autocompletion of the code, indispensable!



```c
/* USER CODE BEGIN 4 */
void blink_toggle(int time, GPIO_TypeDef* GPIOx_PORT, uint16_t LDx_Pin){

    LL_GPIO_TogglePin(GPIOx_PORT, LDx_Pin); // toggle the pin each time you call the function
    LL_mDelay(time); //ms Delay

}
```

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

     /* Uncomment the function that you want to use */

      //blink_toggle(TIME_DELAY, LD2_GPIO_Port,LD2_Pin);   //blink the led using the HAL_GPIO_TogglePin function
      blink_writePin(TIME_DELAY, LD3_GPIO_Port,LD3_Pin); //blink the led using the HAL_GPIO_WritePin function

    /* USER CODE END WHILE */

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

```c
#define LD2_Pin GPIO_PIN_7
#define LD2_GPIO_Port GPIOB
/* USER CODE BEGIN Private defines */
#define TIME_DELAY 1000
/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif
```