



Interrupt with LL Libraries

rev1.0 24/03/2020

GOAL

Generate an interrupt when the blue push button has been pressed

PREREQUISITES

Software needed:

- STM32IDE

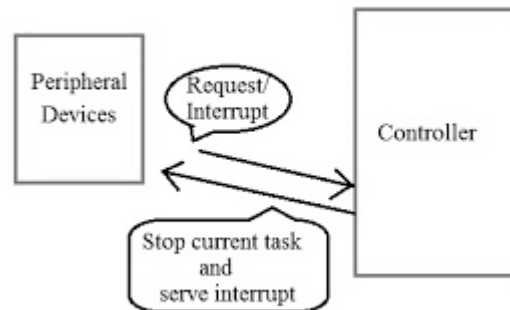
Hardware used in this example:

- **NUCLEO-F446ZE**

What is an Interrupt?

An **interrupt** is an asynchronous signal that allows you to manage situations in which particular attention is needed: there may be events such that it is necessary for the microcontroller to temporarily interrupt the normal execution of the program in order to execute a certain process.

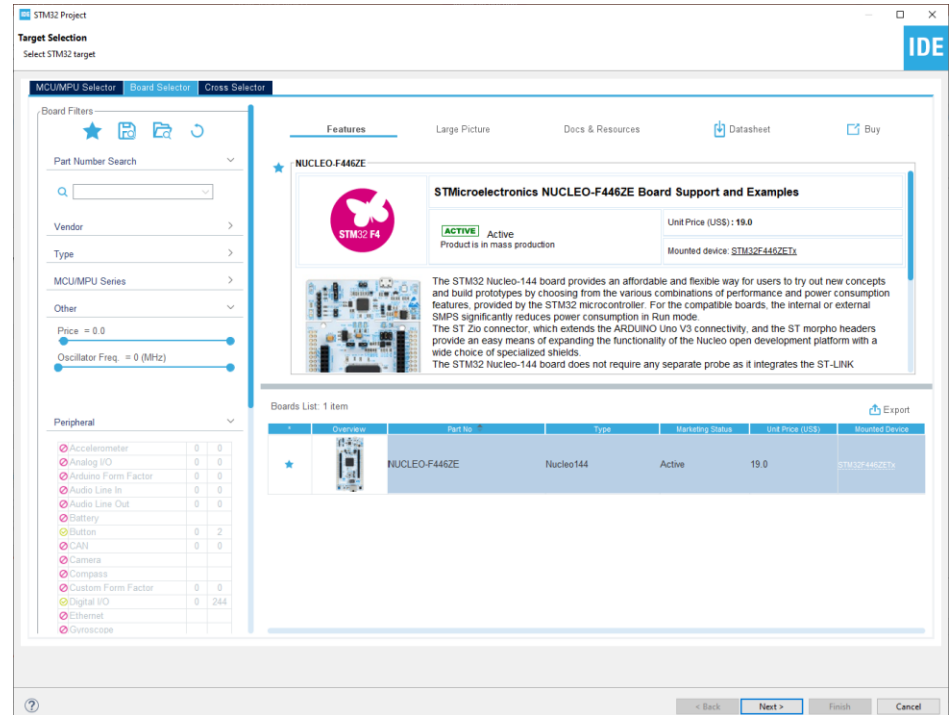
Examples that generate an interrupt may be the pressing of a button, rather than an event at regular time intervals (**timers**) or a certain input being exceeded by a certain threshold.



Start a new project

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

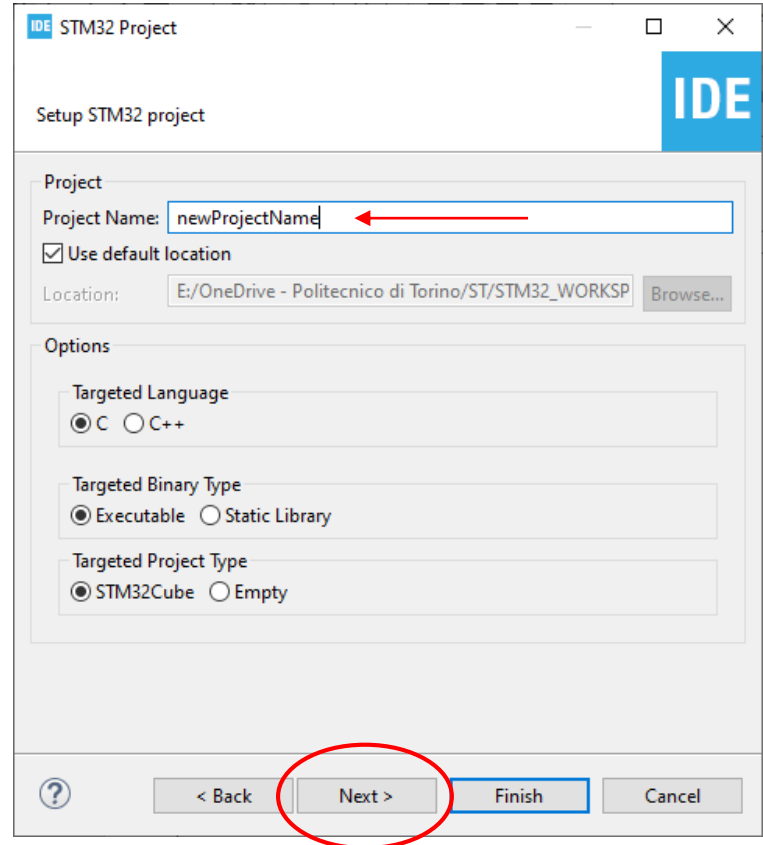
Select your board or your uC and click
next.



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 screenshot shows the 'Setup STM32 project' dialog box in the IDE. The dialog has a title bar with 'IDE STM32 Project' and standard window controls. The main content is divided into sections: 'Project' and 'Options'. In the 'Project' section, the 'Project Name' field contains 'newProjectName' and is highlighted with a red arrow. Below it, the 'Use default location' checkbox is checked. The 'Location' field shows 'E:/OneDrive - Politecnico di Torino/ST/STM32_WORKSP' with a 'Browse...' button. The 'Options' section contains three groups of radio buttons: 'Targeted Language' with 'C' selected, 'Targeted Binary Type' with 'Executable' selected, and 'Targeted Project Type' with 'STM32Cube' selected. At the bottom, there are four buttons: a help button (question mark), '< Back', 'Next >' (circled in red), and 'Finish'. The 'Finish' button is also highlighted with a blue border.

Start a new project

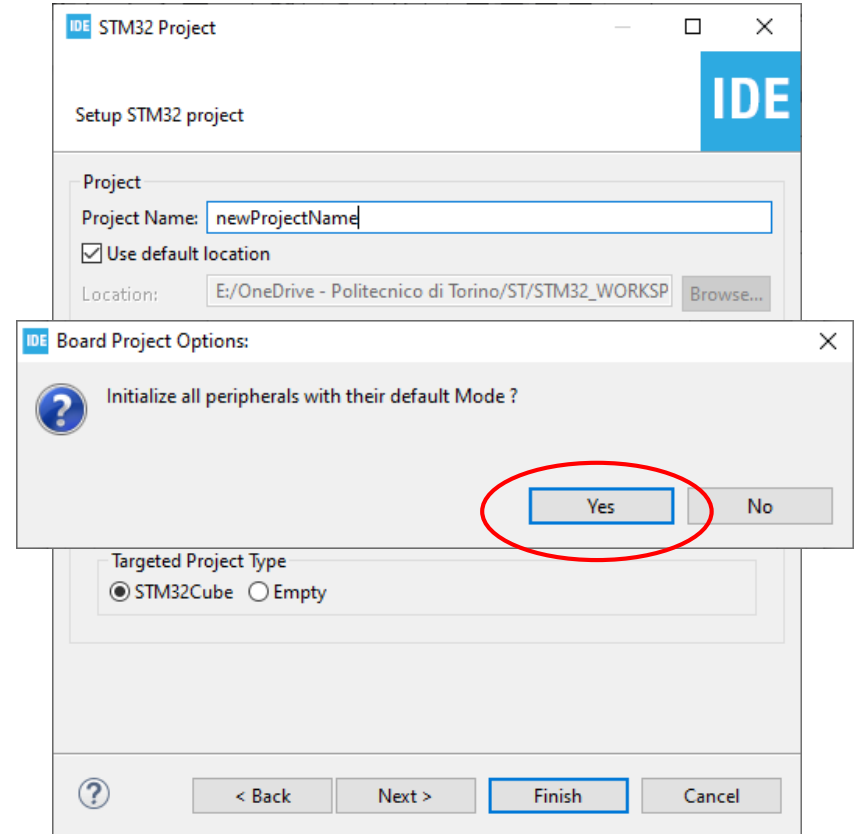
Type the name of your project and click next.

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:

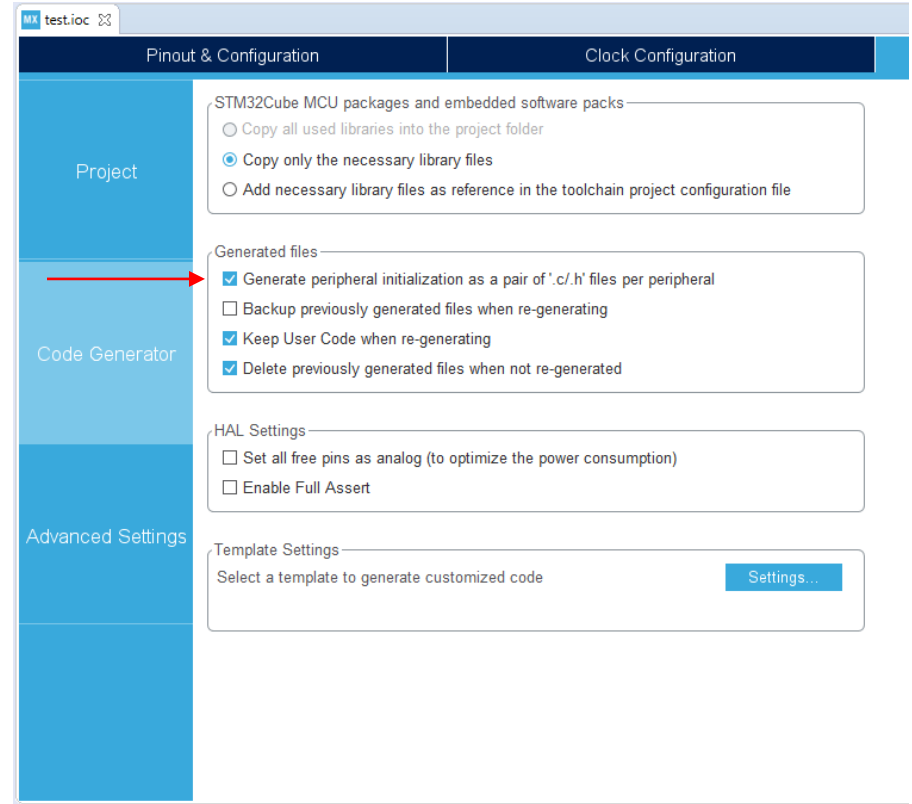
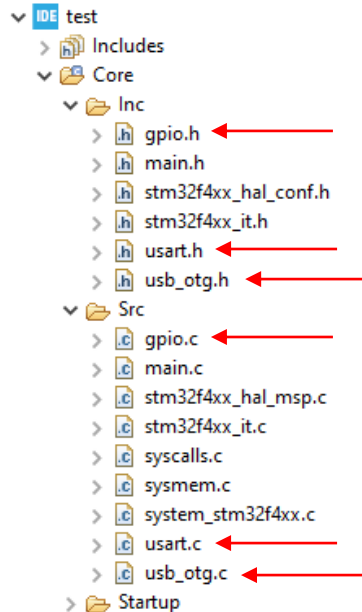
Clicking *Yes* the *USART3*, all the *LEDs* and the blue *UserButton* will be configured as default.

Click *Yes*.



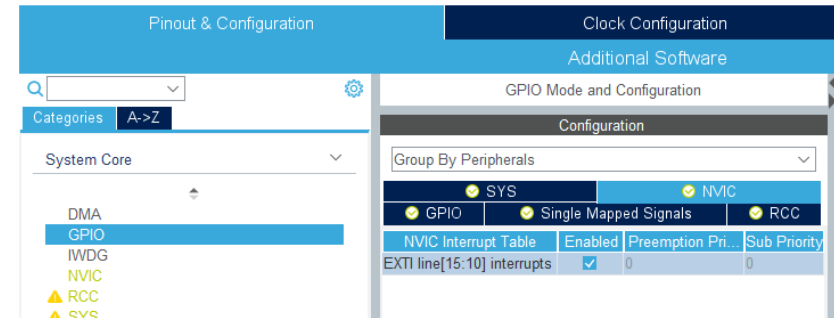
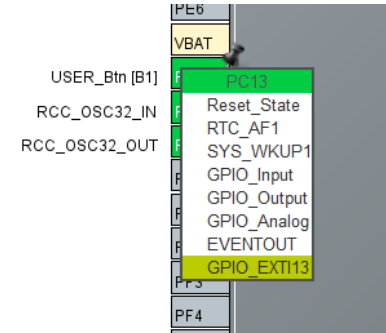
Project Manager

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



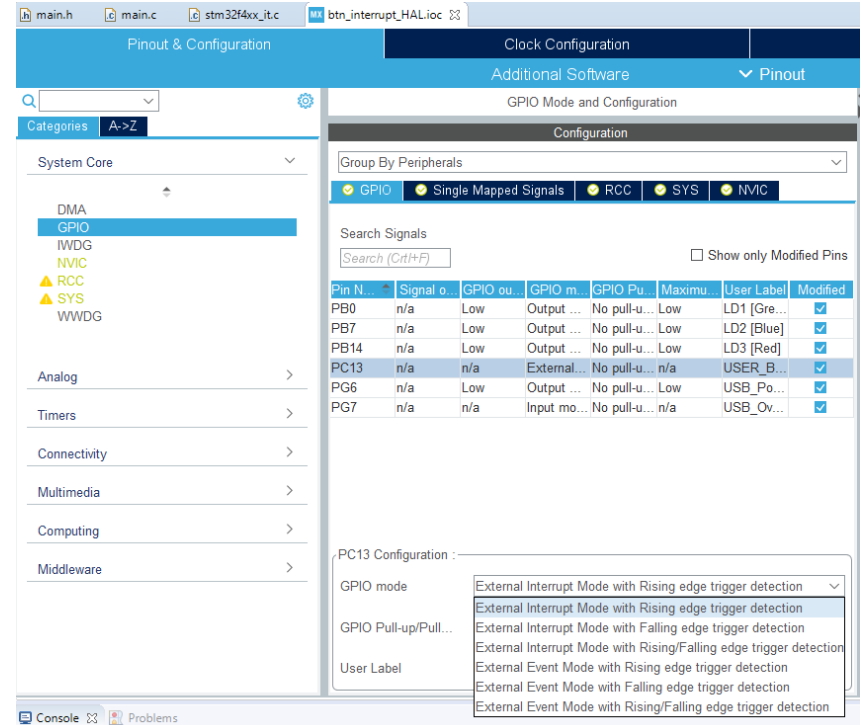
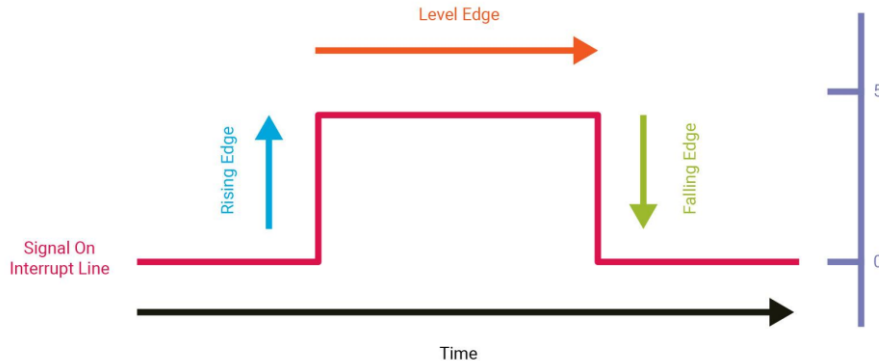
Impostare un GPIO come interrupt

- From *CubeMX* it is necessary to set the *GPIO* pin as **GPIO_EXTIxx**.
- From the **NVIC** (Nested vector interrupt control) Tab we enable the interrupt: this causes *CubeMX* to worry about enabling the interrupt and setting its priority.



Falling & Rising edge detection

From the GPIO card, on the other hand, it is possible to manage the interrupt mode, in particular choosing whether the interrupt should be generated when detecting the rising edge rather than the falling edge, or both.

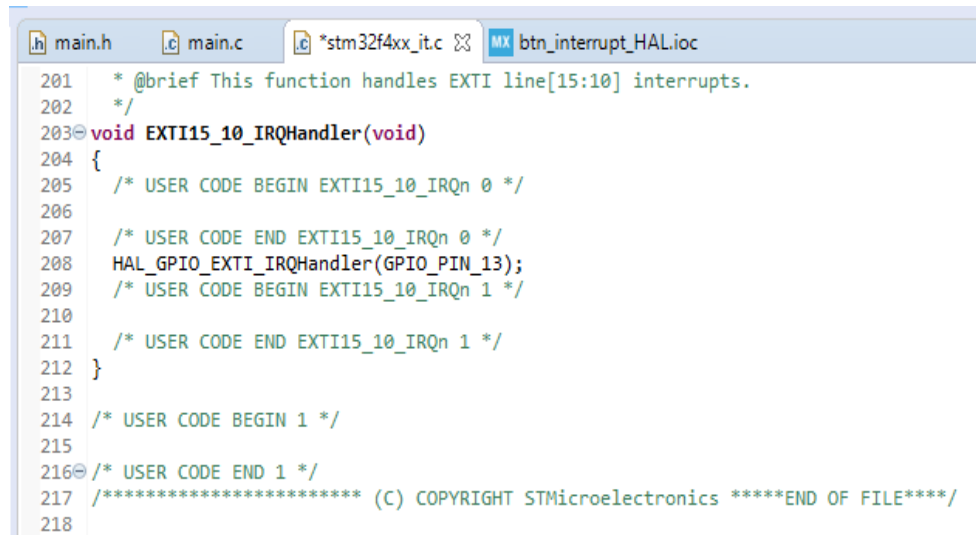


-

At this point we just have to generate the code and start writing our program. In this regard, we want to recreate the previous example but with the difference that this time when the button is pressed an ***interrupt*** will be generated.



- It is useful to note how *CubeMX* has already generated Handler for our interrupt. All Handlers are managed within the ***stm32f4xx_it.c*** file.
- The ***HAL_GPIO_EXTI_IRQHandler()*** will check that the interrupt was generated by the correct line.
- In that case, it will call the *Callback* function that we define in ***gpio.c*** file



```

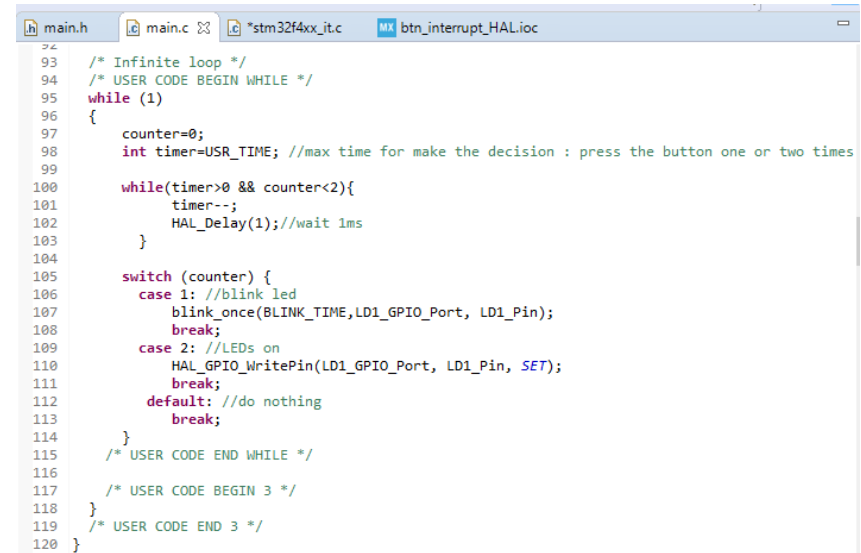
main.h  main.c  *stm32f4xx_it.c  btn_interrupt_HALioc
201  * @brief This function handles EXTI line[15:10] interrupts.
202  */
203  void EXTI15_10_IRQHandler(void)
204  {
205      /* USER CODE BEGIN EXTI15_10_IRQn 0 */
206
207      /* USER CODE END EXTI15_10_IRQn 0 */
208      HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
209      /* USER CODE BEGIN EXTI15_10_IRQn 1 */
210
211      /* USER CODE END EXTI15_10_IRQn 1 */
212  }
213
214  /* USER CODE BEGIN 1 */
215
216  /* USER CODE END 1 */
217  /***** (C) COPYRIGHT STMicroelectronics *****END OF FILE*****/
218

```

Each time the button is pressed, our **counter** variable will then be increased following the generation of the interrupt.



```
242 /* USER CODE BEGIN 4 */
243 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
244     if (GPIO_Pin==USER_Btn_Pin) {
245         //HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
246         counter++;
247     }
248 }
249
```

- Now we just have to slightly modify the code that we wrote in the previous example: the function that took care of reading the button and increasing the counter has been eliminated, this is because now everything is managed through the interrupt.



```
93  /* Infinite loop */
94  /* USER CODE BEGIN WHILE */
95  while (1)
96  {
97      counter=0;
98      int timer=USR_TIME; //max time for make the decision : press the button one or two times
99
100     while(timer>0 && counter<2){
101         timer--;
102         HAL_Delay(1); //wait 1ms
103     }
104
105     switch (counter) {
106     case 1: //blink led
107         blink_once(BLINK_TIME, LD1_GPIO_Port, LD1_Pin);
108         break;
109     case 2: //LEDs on
110         HAL_GPIO_WritePin(LD1_GPIO_Port, LD1_Pin, SET);
111         break;
112     default: //do nothing
113         break;
114     }
115     /* USER CODE END WHILE */
116
117     /* USER CODE BEGIN 3 */
118 }
119 /* USER CODE END 3 */
120 }
```

Debug

- To view the correct management of the interrupt, it may be useful to start the program in **debug** mode , put a breakpoint inside the Callback function and start debugging. 
- When the button is pressed, it will take you to the function that manages the interrupt, as in the next figure.
- At this point it is possible to press resume again or move within the program using the *step* arrows.

