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

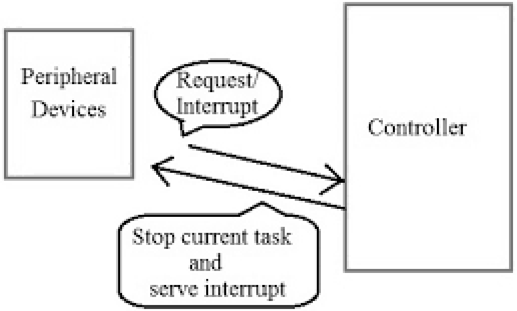
Hardware used in this example:

- **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.

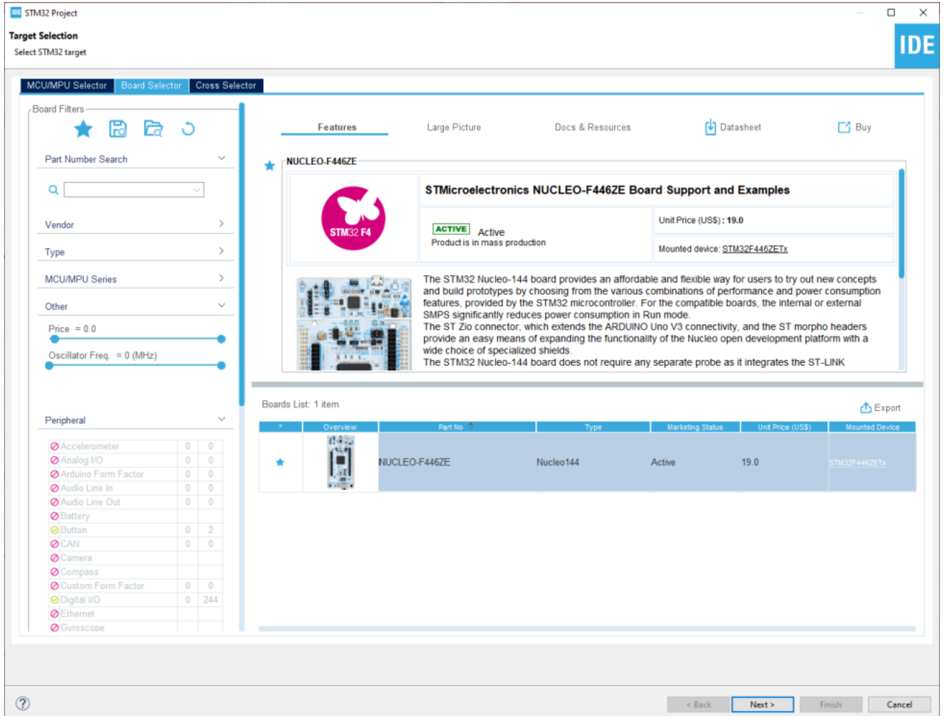
Examples that generate an interrupt may be the pressing of a button, rather than an event at regular time intervals (**timers**) or a certain input being exceeded by a certain threshold.



Start a new project

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

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



Target Selection

Select STM32 target

Price = 0.0

Oscillator Freq. = 0 (MHz)

Accelerometer	0	0
Analog I/O	0	0
Arduino Form Factor	0	0
Audio Line In	0	0
Audio Line Out	0	0
Battery		
Button	0	2
CAN	0	0
Camera		
Compass		
Custom Form Factor	0	0
Digital I/O	0	244
Ethernet		
Gyroscope		

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

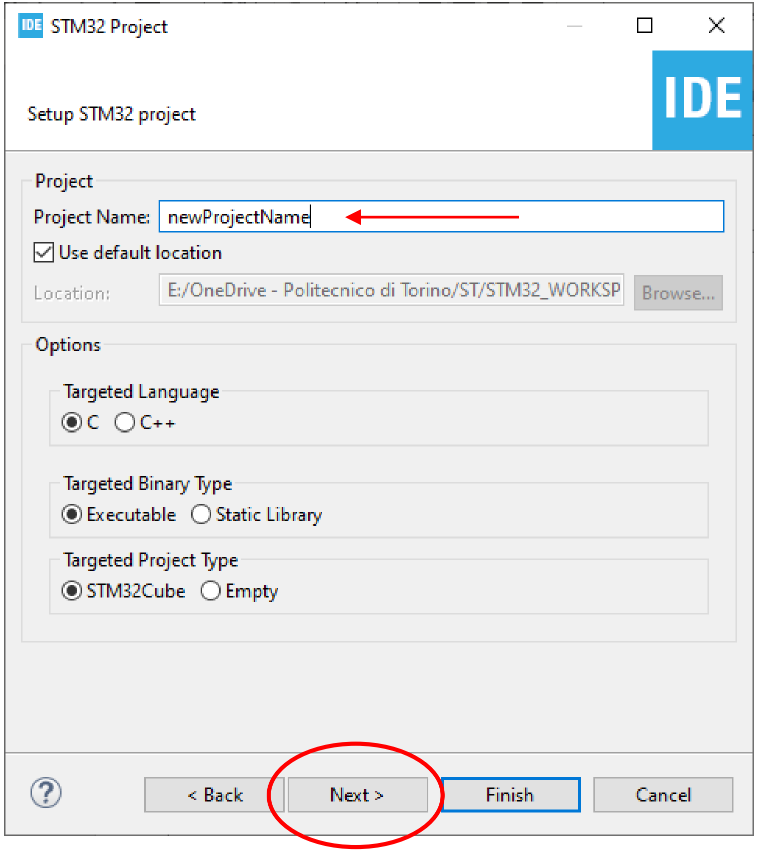
Boards List: 1 item

*	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

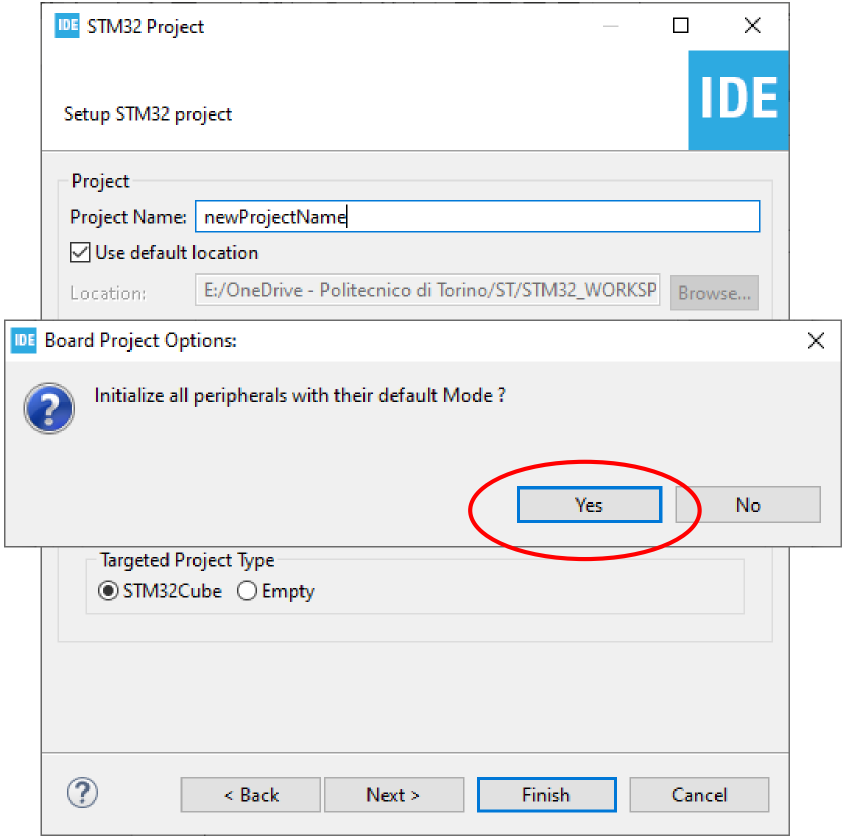
Type the name of your project and click next.

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:

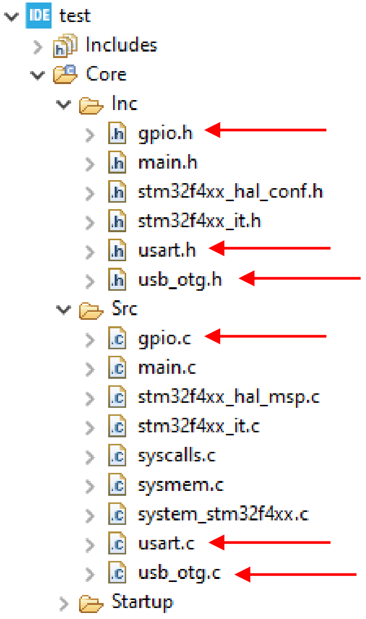
Initialize all peripherals with their default Mode ?

Targeted Project Type

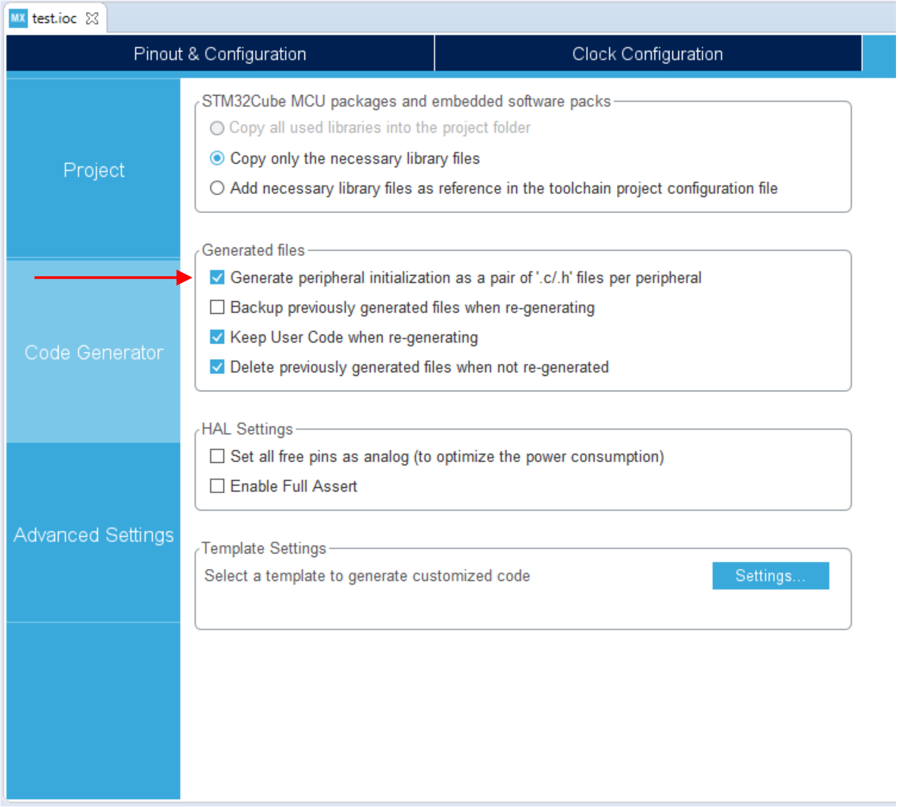
● STM32Cube ○ Empty

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)

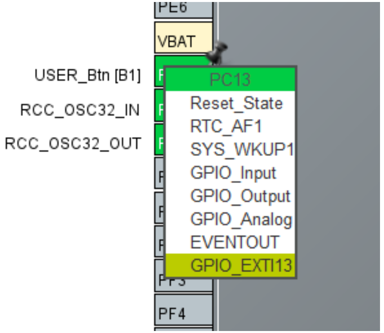
☐ Enable Full Assert

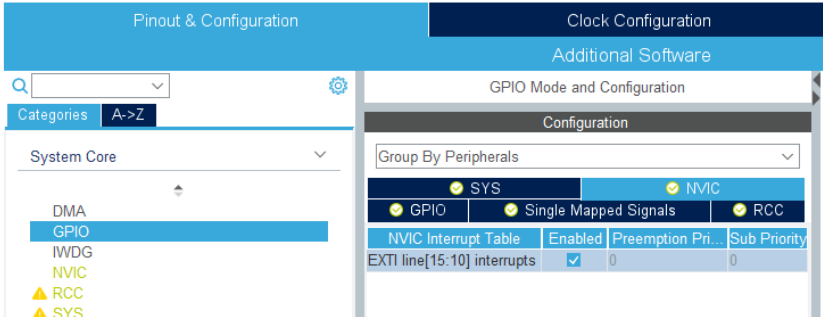
Template Settings

Select a template to generate customized code

Set GPIO pin as 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.

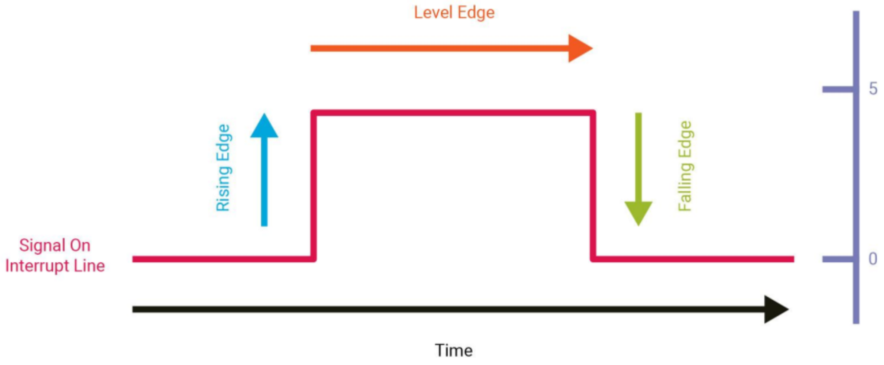


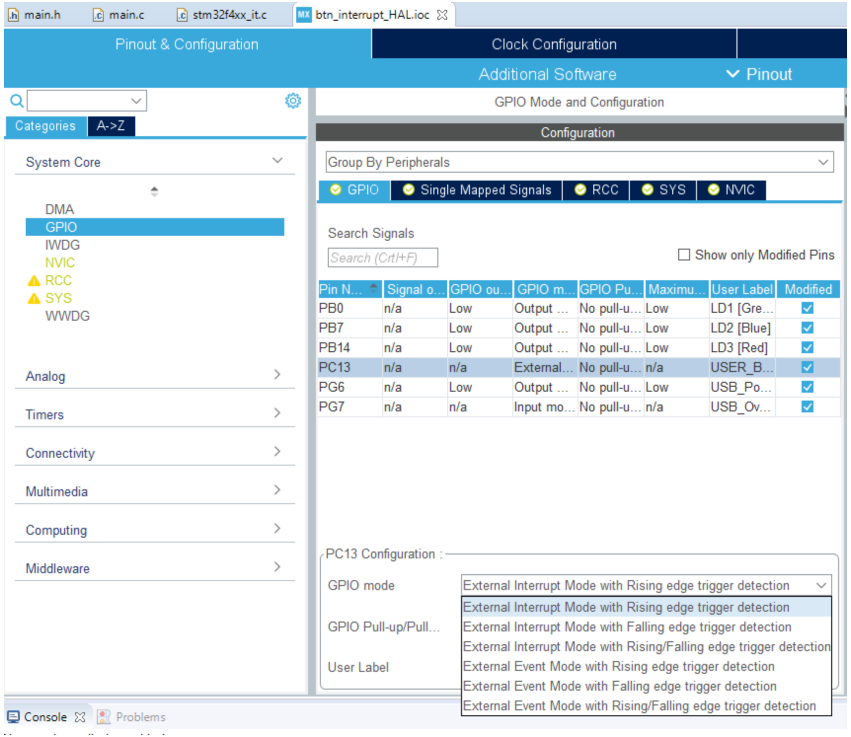


NVIC Interrupt Table	Enabled	Preemption Pri...	Sub Priority
EXTI line[15:10] interrupts	☑	0	0

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.





□ Show only Modified Pins

Pin N...	Signal o...	GPIO ou...	GPIO m...	GPIO Pu...	Maximu...	User Label	Modified
PB0	n/a	Low	Output ...	No pull-u...	Low	LD1 [Gre...	☑
PB7	n/a	Low	Output ...	No pull-u...	Low	LD2 [Blue]	☑
PB14	n/a	Low	Output ...	No pull-u...	Low	LD3 [Red]	☑
PC13	n/a	n/a	External...	No pull-u...	n/a	USER_B...	☑
PG6	n/a	Low	Output ...	No pull-u...	Low	USB_Po...	☑
PG7	n/a	n/a	Input mo...	No pull-u...	n/a	USB_Ov...	☑

PC13 Configuration :

GPIO mode: External Interrupt Mode with Rising edge trigger detection

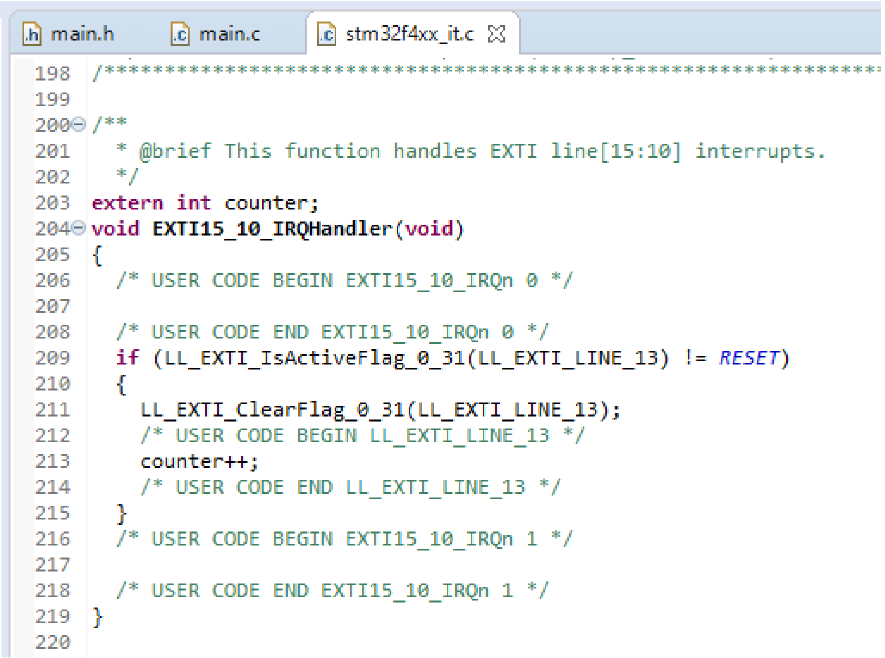
GPIO Pull-up/Pull...

User Label

- External Interrupt Mode with Rising edge trigger detection
- External Interrupt Mode with Falling edge trigger detection
- External Interrupt Mode with Rising/Falling edge trigger detection
- External Event Mode with Rising edge trigger detection
- External Event Mode with Falling edge trigger detection
- External Event Mode with Rising/Falling edge trigger detection

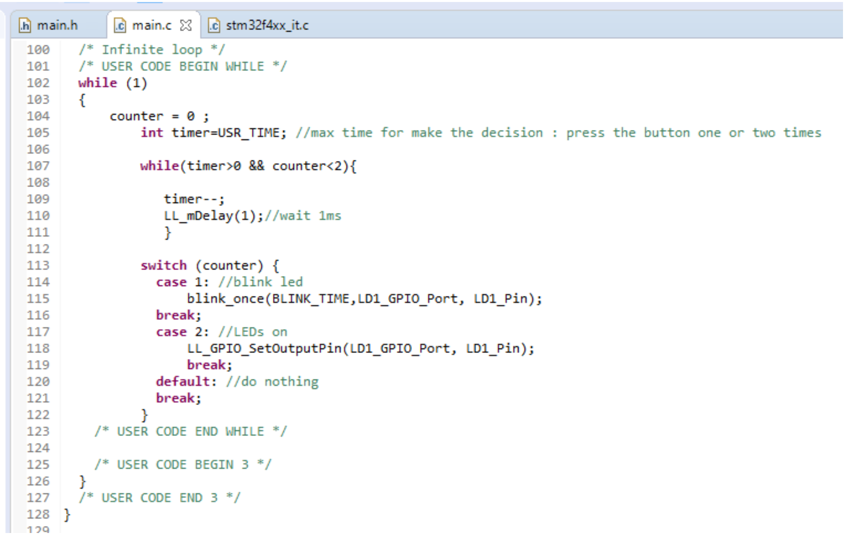
- At this point we just have to generate the code and start writing our program. In this regard, we want to recreate the previous example but with the difference that this time when the button is pressed an ***interrupt*** will be generated.

- 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 ***LL_EXTI_IsActiveFlag()*** function will take care of verifying that the interrupt was generated by the correct line.
- Each time the button is pressed, our counter variable will then be increased. It should be noted that we have called counter as an **extern** variable here, because it was defined within the ***main.c*** file.



```c
/*****************************************************************************

/**
  * @brief This function handles EXTI line[15:10] interrupts.
  */
extern int counter;
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  if (LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_13) != RESET)
  {
    LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_13);
    /* USER CODE BEGIN LL_EXTI_LINE_13 */
    counter++;
    /* USER CODE END LL_EXTI_LINE_13 */
  }
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}

```

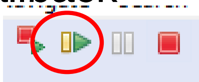
- Now we just have to slightly modify the code that we wrote in the previous example: the function that took care of reading the button and increasing the counter has been eliminated, this is because now everything is managed through the interrupt.



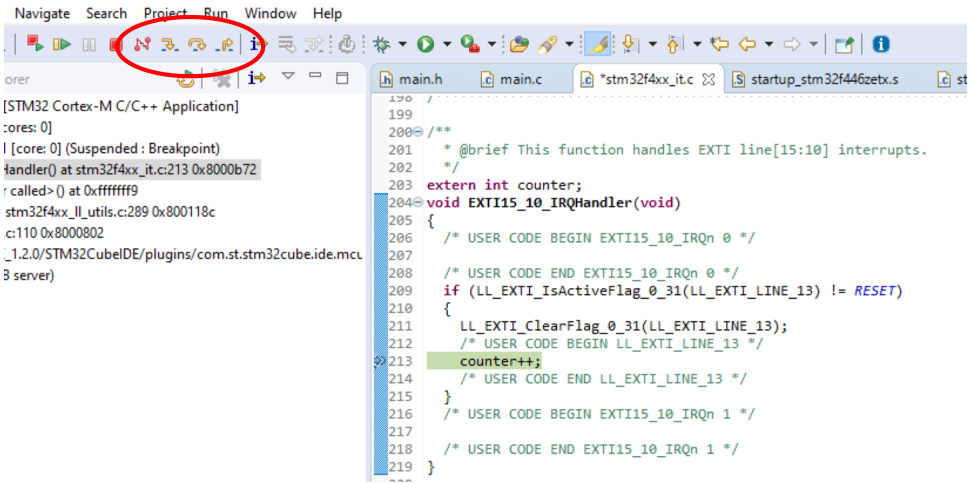
```
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
   counter = 0 ;
     int timer=USR_TIME; //max time for make the decision : press the button one or two times

     while(timer>0 && counter<2){

       timer--;
       LL_mDelay(1);//wait 1ms
       }

     switch (counter) {
      case 1: //blink led
        blink_once(BLINK_TIME,LD1_GPIO_Port, LD1_Pin);
      break;
      case 2: //LEDs on
        LL_GPIO_SetOutputPin(LD1_GPIO_Port, LD1_Pin);
        break;
      default: //do nothing
      break;
     }
  /* USER CODE END WHILE */

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

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.*



```
[STM32 Cortex-M C/C++ Application]
:ores: 0]
l [core: 0] (Suspended : Breakpoint)
Handler() at stm32f4xx_it.c:213 0x8000b72
called>() at 0xfffffff9
stm32f4xx_ll_utils.c:289 0x800118c
c:110 0x8000802
_1.2.0/STM32CubeIDE/plugins/com.st.stm32cube.ide.mcu
3 server)
```

```
/**
  * @brief This function handles EXTI line[15:10] interrupts.
  */
extern int counter;
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  if (LL_EXTI_IsActiveFlag_0_31(LL_EXTI_LINE_13) != RESET)
  {
    LL_EXTI_ClearFlag_0_31(LL_EXTI_LINE_13);
    /* USER CODE BEGIN LL_EXTI_LINE_13 */
    counter++;
    /* USER CODE END LL_EXTI_LINE_13 */
  }
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}
```