GPIO Interrupts (EXTI) on STM32 Microcontrollers using HAL
with FreeRTOS enabled

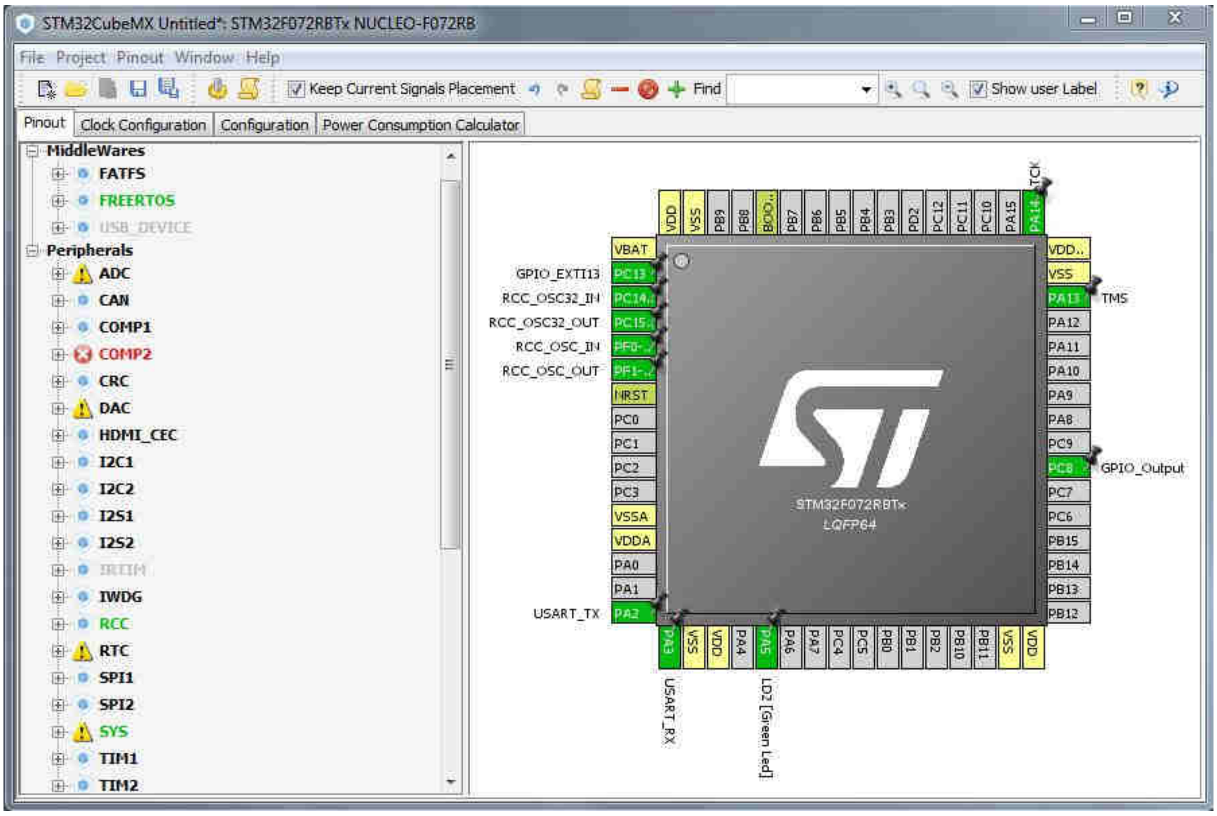
The STM32 microcontroller family offers multiple GPIO interrupt pins. The STM32CubeMX Software comes in handy when configuring the parameters of these pins. However, the actual usage of

This tutorial uses the following equipment:
- NUCLEO-F072RB Board
- Keil uVision 5 with the necessary packages for Nucleo boards installed
- STLink USB Driver
- STM32CubeMX
- External LED on PC8 (to demonstrate that the endless loop is not affected)

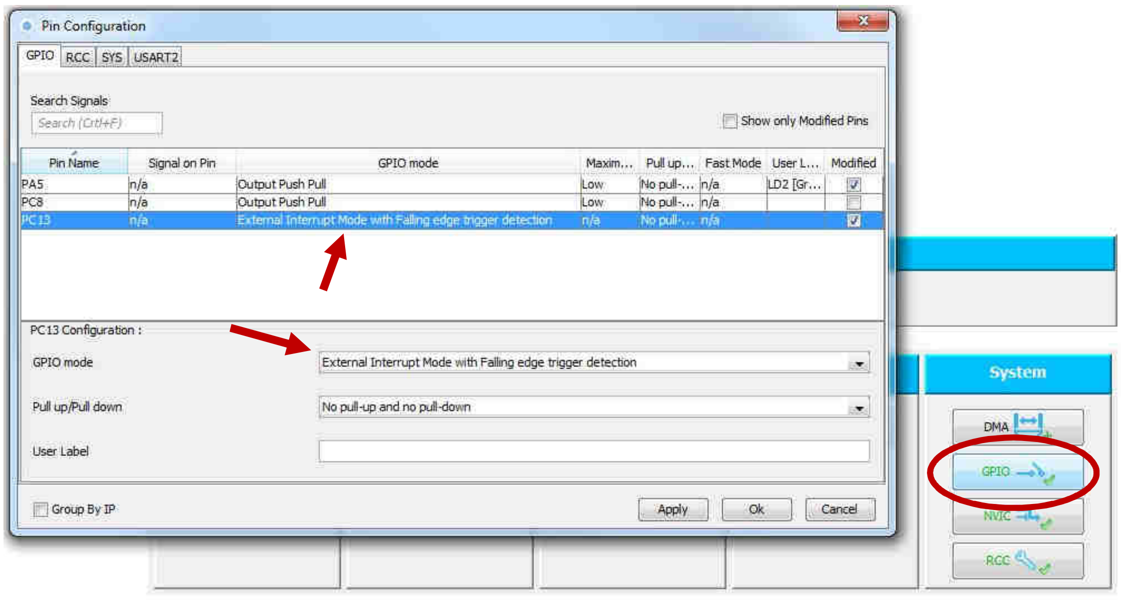
STM32CubeMX

Generating the config. files from STM32CubeMX.

1. Open STM32CubeMX and open a new project.
2. Select the Nucleo-F072RB from the Borards tab
3. Enable FreeRTOS
4. Set the RCC (HSE & LSE) to Crystal/Ceramic Resonator
5. Enable the USART2 port in Asynchronous mode
6. (optional) set any GPIO pin as output (and connect an LED to it, here PC8 is used)
7. Set PC13 as GPIO_EXTI13 (this is the blue user button on the Nucleo)



8. Click the configuration tab and click on the GPIO button
9. Here you can set the interrupt parameters for EXTI13
   I set the GPIO mode to detect falling edges (since the user button has a pull-up resistor)
   Hit Apply and Ok to save the changes.



Pin Name	Signal on Pin	GPIO mode	Maxim...	Pull up...	Fast Mode	User L...	Modified
PA5	n/a	Output Push Pull	Low	No pull-...	n/a	LD2 [Gr...	✓
PC8	n/a	Output Push Pull	Low	No pull-...	n/a		
PC13	n/a	External Interrupt Mode with Falling edge trigger detection	n/a	No pull-...	n/a		✓

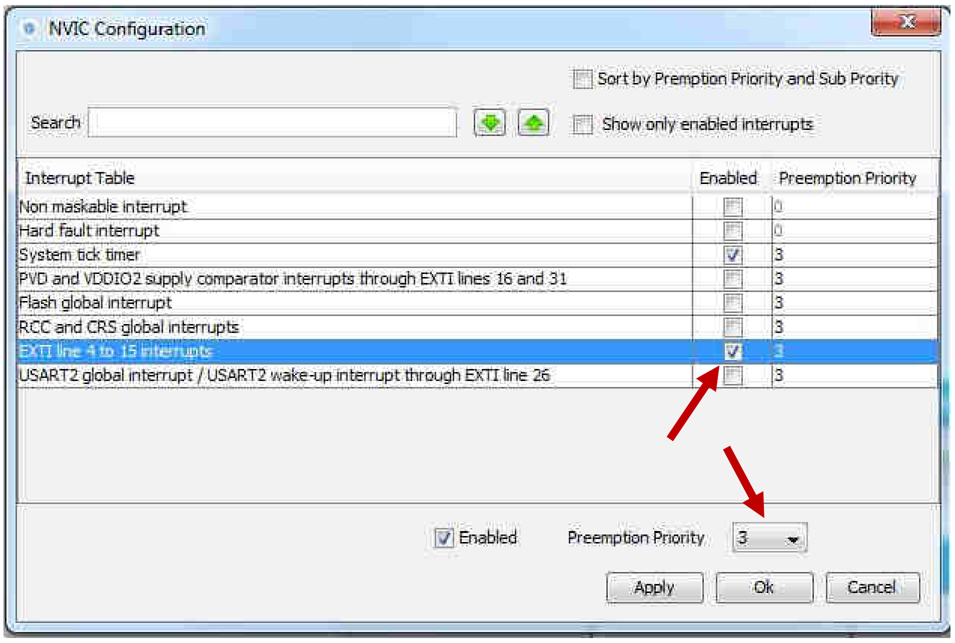
PC13 Configuration :

GPIO mode: External Interrupt Mode with Falling edge trigger detection

Pull up/Pull down: No pull-up and no pull-down

User Label:

10. Next, click the NVIC button
11. Enable EXTI lines 4 – 15 (since we use EXTI13)
12. The priority is 3 by default, this is because we are using FreeRTOS


NVIC Configuration
Sort by Premption Priority and Sub Prority
Search
Show only enabled interrupts
Apply
Ok
Cancel

Interrupt Table	Enabled	Preemption Priority
Non maskable interrupt		0
Hard fault interrupt		0
System tick timer	✓	3
PVD and VDDIO2 supply comparator interrupts through EXTI lines 16 and 31		3
Flash global interrupt		3
RCC and CRS global interrupts		3
EXTI line 4 to 15 interrupts	✓	3
USART2 global interrupt / USART2 wake-up interrupt through EXTI line 26		3

✓ Enabled Preemption Priority 3

13. Go to Project > Generate code
14. Enter a project name and select MDK-ARM V5
15. Generate the code and open the project in Keil uVision

The code generator should output the desired code as usual.
Note, that the EXTI initialisation takes part within the GPIO init function call.

```
204   /*Configure GPIO pin : PC13 */
205   GPIO_InitStruct.Pin = GPIO_PIN_13;
206   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
207   GPIO_InitStruct.Pull = GPIO_NOPULL;
208   HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
209
```

On line 206, the GPIO mode is set to detect interrupts on falling edges.

```
/* EXTI interrupt init*/
HAL_NVIC_SetPriority(EXTI4_15_IRQn, 3, 0);
HAL_NVIC_EnableIRQ(EXTI4_15_IRQn);
```

This initialisation is found at the end of the GPIO init.

Set a default task in an endless loop

So we can test that the interrupt is working. Type the following into the endless loop of the default task.

```
/* USER CODE BEGIN 5 */
/* Infinite loop */
for(;;)
{
  GPIOC -> ODR ^= GPIO_PIN_8;    // toggle the LED on & off
  osDelay(500);
}
/* USER CODE END 5 */
```

This simply shows, that the endless loop keeps running when we trigger an EXTI interrupt.

Do something on an interrupt event

Open the file stm32f0xx_it.c
That is where the magic happens.
Somewhere on line 73 should be a function called `EXTI4_15_IRQHandler`. Each time an EXTI interrupt is triggered, this function gets called. The statement between the two user code segments simply clears the interrupt flag of the EXTI channels.

Now to execute something on an EXTI interrupt event, you could simply add your code between the user code blocks. This code is then executed with every EXTI event on any enabled EXTI channel.

Filter EXTI interrupt events

But you might want to do different things, when different interrupt pins are triggered.
You can filter the interrupt event with a simple if-statement (see on line 76).
It is important to place this if statement before clearing the interrupt flags (line 80).

```c
73  void EXTI4_15_IRQHandler(void)
74  {
75    /* USER CODE BEGIN EXTI4_15_IRQn 0 */
76    if(__HAL_GPIO_EXTI_GET_FLAG(GPIO_PIN_13)){
77      LD2_GPIO_Port -> ODR ^= LD2_Pin;          // toggle LD2 LED
78    }
79    /* USER CODE END EXTI4_15_IRQn 0 */
80    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
81    /* USER CODE BEGIN EXTI4_15_IRQn 1 */
82
83    /* USER CODE END EXTI4_15_IRQn 1 */
84  }
85
```

The instructions within the if-statement is only executed when an EXTI interrupt on EXTI channel 13 occurs.

Keep the following makro in mind when filtering EXTI interrups:
It takes the EXTI channel as an argument, where x is any numer from 0 to 15.
It returns true, when the interrupt flag for the given channel is set.

```c
__HAL_GPIO_EXTI_GET_FLAG(GPIO_PIN_x)
```

Test the program

If you compile and upload the code, the LED connected to PC8 should blink independently of the user button. With every press of the user button, the LD2 LED on the Nucleo board should toggle.

Document Created by Simon Burkhardt

Simon Burkhardt
electronical engineering
simonmartin.ch

History:

V1.0	tested the code, created this document
V1.0.1	added contact info and cc0 notice