twoofanerd

Straigh to the code!

Microcontrollers

Easy interrupt handling on STM32, the quick tutorial

17/06/2020 Giuseppe 0 Comments

In this post we will introduce the interrupt handling topic, as easy as we can, on STM32 boards. Interrupt can be seen as an event which causes a deviation from the regular program flow. This kind of event can occur anytime, especially at the same time the microcontroller is busy doing other operations. On interrupt arrival, program flow is "paused" and a particular function is called. When this event handling function ends, the main program flow is resumed back.

Search

Archives

June 2020

Categories

Microcontrollers

Various

Meta

Log in

Entries feed

Comments feed

WordPress.org

Knowing more: these particular functions which are called on interrupt events are know as ISR (Interrupt Service Routines). Their signature and name are predetermined and they are never called from the user. When ISR's are present in the code and interrupt is configured, they are automatically called from the interrupt controller. Main program flow is temporarily interrupted, just for the time to complete the handling routine.

1. Why we need interrupts

Two main solutions exist, when dealing with this kind of "external world" interfacing problems. First one is called "polling" while second is interrupt based.

Polling is the solution we adopted in the previous example. Polling a status means continuously asking for it, observing changes by using our code. Interrupt is the inverse approach, the peripheral informs us when a change has occured, without the need, for us, to continuously request the status.

3. Are interrupts really needed?

Interrupt handling has advantages and drawbacks but sometimes interrupts are the only viable solution. We will create a specific example, starting from our very first project or from the previous tutorial. This time we will write a small piece of code able to toggle the led everytime the pushbutton is pressed. Toggling will happen when the key goes from released to pressed. Remember that B1 is normally high (1 = SET), as it is connected to the power supply by default. With the following code, the led will start from the on status. We are always working on the main.c source file: first code block must be copied inside the PV block, the other goes inside the WHILE block, as indicated.

```
[... This block goes in the initial part of main.c
near line 46 ...]
/* USER CODE BEGIN PV */
//Declaring three globals
GPIO_PinState previousStatus = GPIO_PIN_RESET;
```

```
GPIO_PinState currentStatus = GPIO_PIN_RESET;
GPIO_PinState ledStatus = GPIO_PIN_RESET;
/* USER CODE END PV */

[... This block goes in the middle part of main.c
around line 99 ...]

  /* USER CODE BEGIN WHILE */
  while (1)
  {
          //Read the current button pin status
          currentStatus =
HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);

          //An action must be taken if the button went
from low to high
          if((currentStatus == GPIO_PIN_SET) &&
(previousStatus == GPIO_PIN_RESET))
          {
                  //If led was off
                  if(ledStatus == GPIO_PIN_RESET)
                  {
                          //I'll power it on
                          ledStatus = GPIO_PIN_SET;
                  }//Otherwise if it was on
                  else if(ledStatus == GPIO_PIN_SET)
                  {
                          //I'll power it down
                          ledStatus = GPIO_PIN_RESET;
                  }
          }
          //Copy back current status to previous
          //Next loop it will represent the "past"
status
          previousStatus = currentStatus;

          //After all the calculations, copy the
variable to the
          //Real led
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin,
ledStatus);
    /* USER CODE END WHILE */

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

This example works perfectly as our microcontroller is only dedicated to this task. Loop duration is times smaller the time we need to press and release the button. The microcontroller will have no problems in catching even your fastest clicks. Problems start arising when our algorithm becomes more complex and time consuming. As the loop will be looser, it will be easier for the microcontroller to skip some of your clicks on the button. This will cause also missing toggles for the led.

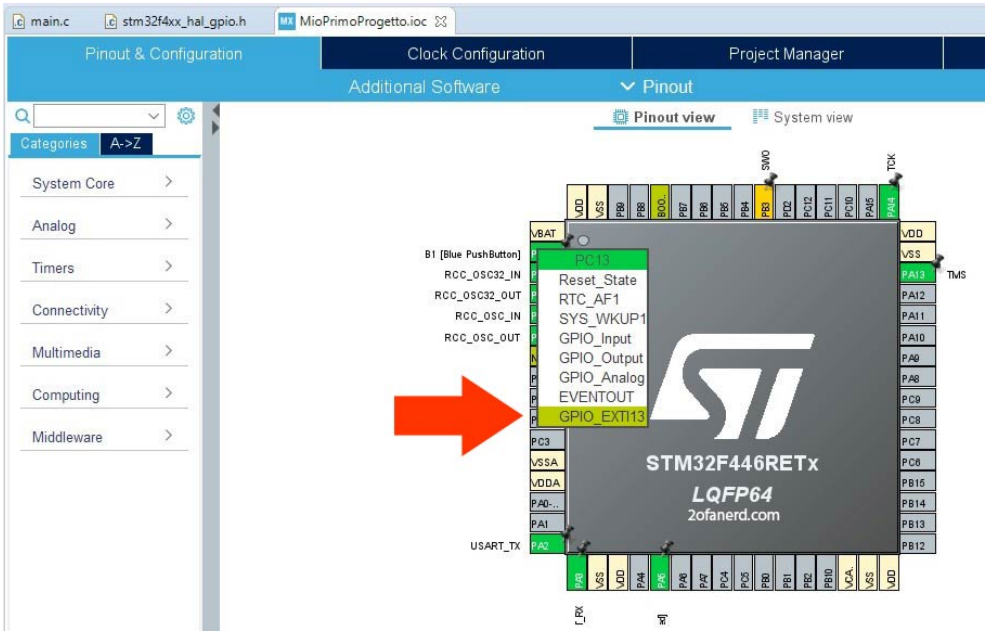
To simulate the “complex task” which is time demanding, we are introducing an half second delay just after the led actuation line of code. Our code will become:

```
[... Same code as before ...]
          //After all the calculations, copy the
variable to the
          //Real led
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin,
ledStatus);
          //My very complex and demanding task
          HAL_Delay(500);
    /* USER CODE END WHILE */

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

The microcontroller now has the opportunity to “see” the button status only two times per second. Toggling the led status precisely will become more difficult than before. The solution to this problem comes introducing the interrupt technique.

3. Interrupt activation

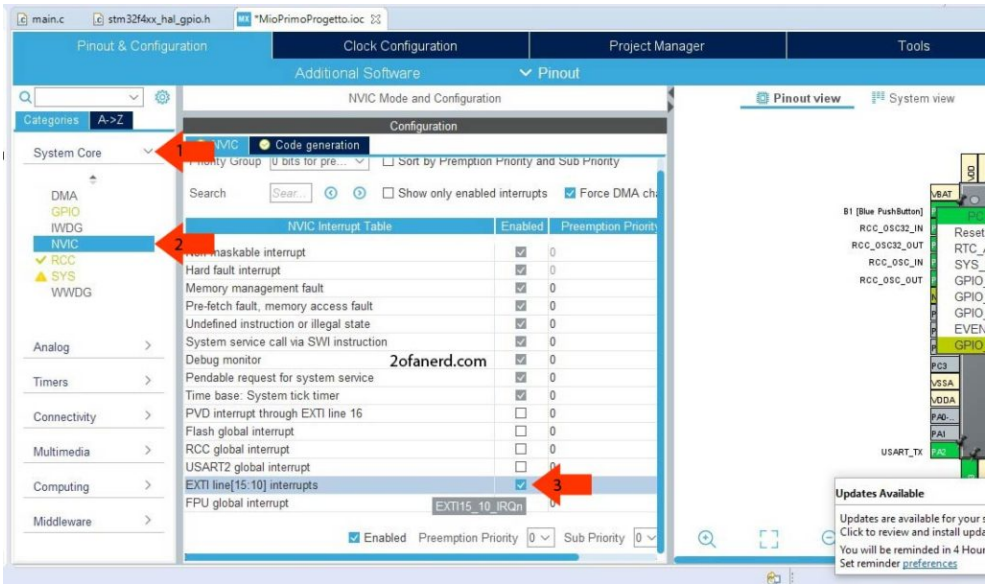
Open the visual configuration editor for the microcontroller, by double clicking on the .ioc file (the blue MX icon file). Interrupt activation starts from hardware configuration of some aspects. First of all, look closely at the microcontroller pushbutton pin configuration. You will notice that the pin is in EXTI mode (EXTI stands for EXTernal Interrupt). When a pin is configured as EXTI it works both as digital input (like we did in the previous example, without even noticing) and as interrupt generation “source” when its status changes. Notice also that the pin is connected to the interrupt line 13 (EXTI13).



Pin was already set as GPIO_EXTI, the number 13 is the interrupt line identifier.

> **Knowing more:** microcontroller integrates an interrupt controller, called NVIC. The NVIC manages events which can generate interrupts and their priority. Every microcontroller has limited number of interrupt lines. Interrupts can also be generated from other “events”, for example the arrival of data on the serial peripheral. We will deal with these scenarios in more advanced tutorials.

Now we have to write the “special function”, which has to be called automatically on the interrupt event. Always using the graphical editor click on “System Core”, then on “NVIC” and enable interrupt checkbox on lines EXTI 10 to 15, as we need triggering on line 13.



Enable interrupts on line 15:10 (from 10 to 15) as we need it on line 13.

> **Knowing more:** we are enabling line 13 interrupt management on the NVIC (Nested Vectored Interrupt Controller) which is a special component in charge for the interrupt handling. Remember: when interrupt is triggered, a "special" function is called.

Now we need to add a new function to our main.c source file, which is the handler for the interrupt. This function has a precise name and signature which cannot be changed. Outside our main function, let's look for a /* USER CODE BEGIN x */ and /* USER CODE END x */ to add the following block. We have chosen the USER CODE BEGIN/END 4 block.

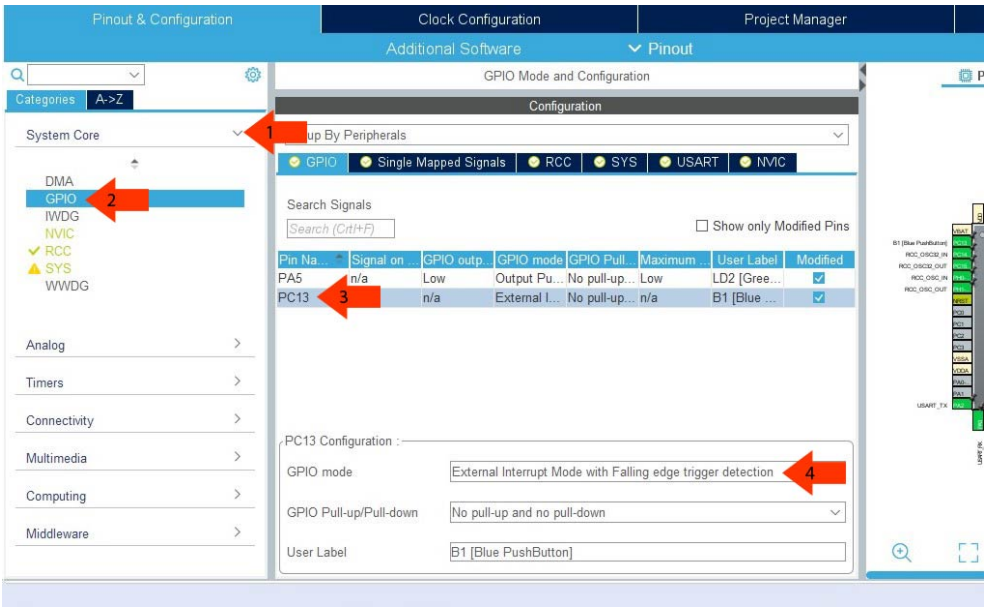
```
/* USER CODE BEGIN 4 */
//This function is automatically called from the
interrupt
//Its name and signature are fixed
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        //This function is called for every EXTI event
so
        //we need first to sort out which pin sourced
it
        if(GPIO_Pin == B1_Pin)
        {
          //Il Led was off
          if(ledStatus == GPIO_PIN_RESET)
```

```
          {
                  //Power on
                  ledStatus = GPIO_PIN_SET;
          }//If was on instead
          else if(ledStatus == GPIO_PIN_SET)
          {
                  //Power off
                  ledStatus = GPIO_PIN_RESET;
          }

          //Finally the variable is actuated on the
led
          HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin,
ledStatus);
        }
}
/* USER CODE END 4 */
```

Now, our main loop can only concentrate on the “demanding task”, which now is a simple delay. Our loop will become:

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
        //Main has only to care about the "demanding
task"
        HAL_Delay(500);
    /* USER CODE END WHILE */

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

We got an extraordinary result: our microcontroller is concentrating its computing power only on the demanding task, now simulated, without caring about the pushbutton. When the button is pushed, the interrupt is raised and the specific handling function automatically called. We have no longer to care about the previous state of the button, as this is also carried by the edge detection feature of the interrupt controller. The edge direction (rising or falling) is configured also from the graphical configuration editor.



From the list menu you can choose which direction has to follow the input on the pin to trigger the interrupt. In our example falling edge is hardwired with the button push.

> **Warning:** this tutorial is an introduction. You must know that code inside interrupt handling routine has some sort of limitations. One of them is related to the complexity and duration. Another important one is related to the variable sharing between main loop and interrupt routine. We will take our time understanding these aspects in more advanced tutorials.

4. Summary

Interrupt handling is always a difficult topic, but we are sure that you have really caught the core. There are some other things to know but, for now, you can skip them and do some experiments with the previous code. Now your algorithm can go straight without polling digital pin statuses and concentrate all the computing power on your automation task. When the right event occurs, the interrupt is fired, your handling function is called and the “exceptional” situation is managed in the code.

See you soon!

← Reading digital inputs easily on STM32

Giuseppe

Computer science, automation and control of complex system engineer. Full-time developer in firmware, field process, low level and also full-stack web. Computer science, electronics, radio-ham entusiast.

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment