April,2023 Semester

# ICT 6641
# Advanced Embedded System Design

## Lecture#3:
## Inputs and External Interrupts

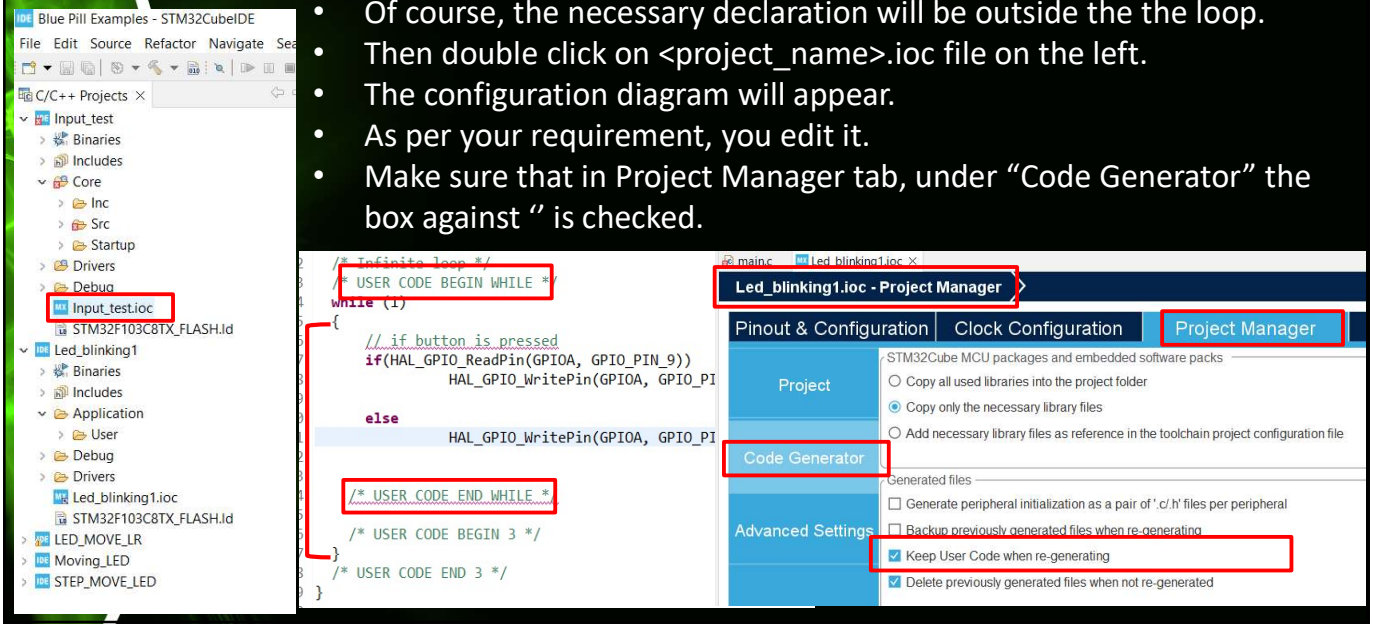
S. M. Lutful Kabir, *PhD*
Professor, BUET

## Today's Topics

- Topic#1: How to update the configurations and update the corresponding code.
- Topic#2: How to download program in the IC.
- Topic#3: How to take input from outside.
- Topic#4: How to configure and handle External Interrupts.

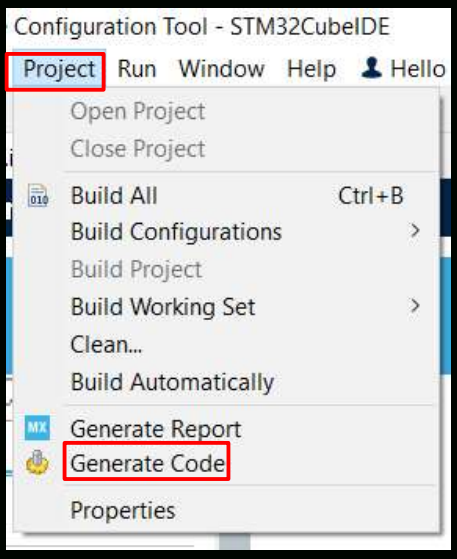
# Updating configuration

- First all make sure that your code is within the while loop boundaries.
- Of course, the necessary declaration will be outside the the loop.
- Then double click on <project_name>.ioc file on the left.
- The configuration diagram will appear.
- As per your requirement, you edit it.
- Make sure that in Project Manager tab, under “Code Generator” the box against ‘’ is checked.




# Updating Configuration (continued)

- Then generate the code again. Project->Generate code
- You will observe that your changes have been reflected in the proper functions and your code remained as it was.
- Let us do an example.
- Let us say, the LED blinking module has to be updated for changing the port pin. How to do it?



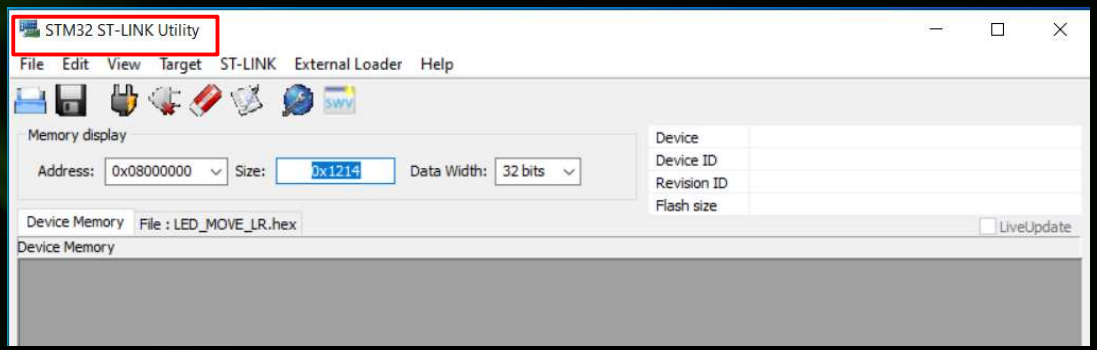
# Demonstration#1

Practical Demo:

- Project Name > LED_blinking
- Let us have the same code that we discussed in the last class (blinking a LED).
- Let us connect the LED in a different pin (from A8 to C13) and regenerate the code.
- Check whether the MX_GPIO_INIT() has been changed and previous user code (while loop and associated definitions of variables/functions, if there were any) remained unchanged.
- Change the code and also the proteus diagram for new pin assignment.
- Re-run the blinking simulation in Proteus and note the effect

# How to download a program into our microcontroller from STM32 Cube IDE

- We shall use ST-Link module for downloading program from STM32 Code IDE.
- For that purpose we have to down load a software name ST-Link Utility from st.com web site.
- After installing, we have to run the program.
- The home screen is as follows.



# Demonstration#2

- Connect terminals of ST-Link dongle with the microcontroller.
- Connect the ST-Link dongle with the laptop/computer using one USB port.
- Run ST-Link utility.
- From File menu open the hex file of the program of demo#1 into Blue Pill module.
- From Target menu run Program.
- Observe the effect.

# Taking Input from outside

- From the graphical interface you may define a pin as input.
- You have the liberty to add PULL-UP or PULL-DOWN resistor and also you may not connect neither of them.
- All other steps are same as defining a pin as output (that we learned in the last class).
- Let us develop the following example and learn how to take input and use it for a purpose.
- Say, there are eight LEDs connected to PA0 to PA7, an input (using a push button) is taken from a pin, say PB1.
- One of the LED will glow.
- When the input is HIGH, the glowing LED will move from left to right, otherwise right to left.
- Using STM32 Cube IDE develop a project named “PUSH-BUTTON”.

## Demonstration#3

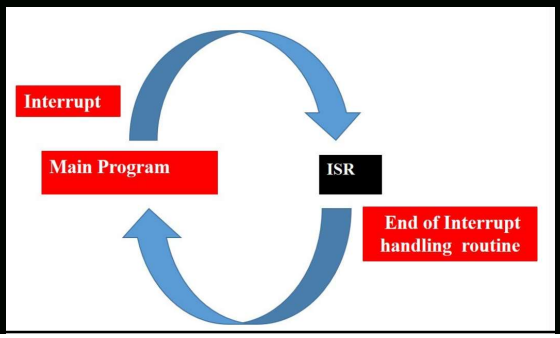
- Following the steps described in demo#2, load the program of “input” in the Blue Pill module.
- Observe that the output is as programmed.
- Identify the bouncing problem in pressing the push switch.
- Suggest the solutions and apply any one of them.
- Observe the effect.

## Introduction to interrupt

- Interrupts are used to handle events that do not happen during the sequential execution of a program.
- Normally a task executes sequentially in our program.
- But there are few tasks that only execute when a special event occurs such as an external trigger signal to the digital input pin of a microcontroller.
- An external interrupt or a ‘hardware interrupt’ is caused by the external hardware module.
- For example, a touch is detected when somebody touches a touch sensor which is connected to a GPIO pin.
- Now, we will focus on this type of external interrupts. There are also some software interrupts that will be discussed later.

## Working Process

- With interrupt, we do not need to continuously check the state of the digital input pin.
- When an interrupt occurs (a change is detected), the processor stops the execution of the main program and a function is called upon known as ISR or the Interrupt Service Routine.
- The processor then temporarily works on a different task (ISR) and then gets back to the main program after the handling routine has ended.
- This is shown in the figure.




## An Example Scenario

- Say a push button is to start a motor.
- A push button can be used to trigger an interrupt.
- Therefore, when an external event occurs, the processor stops what it is doing.
- Then the processor executes the interrupt service routine which we define for the respective event.
- Let us assume that in the routine the code is written for starting the motor.
- After that, it returns to the current position of the main program.
- Similarly there could be a separate switch for stopping the motor.
- External Interrupts are extremely useful because with their help we do not have to constantly monitor (called polling) the digital input pin state.

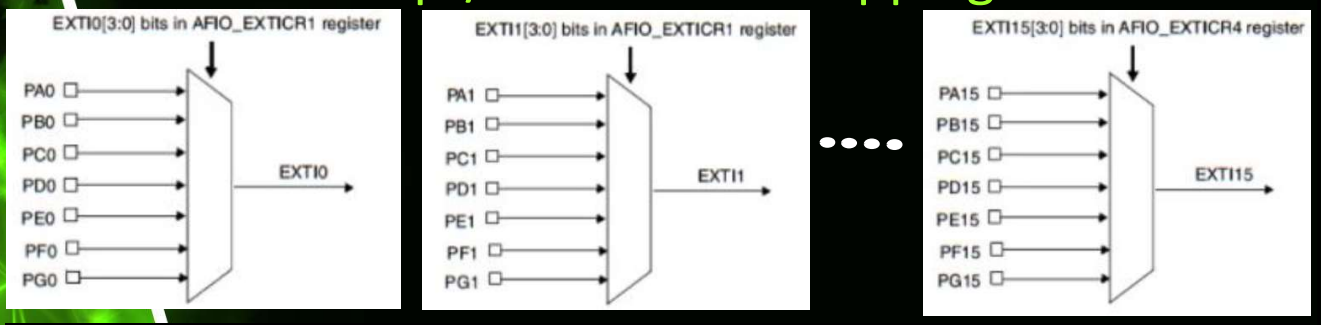
# ARM Cortex Interrupts

- The STM32 ARM microcontroller interrupts are generated in the following manner:
  - The system runs the ISR and then goes back to the main program.
  - The NVIC and EXTI are configured.
  - The Interrupt Service Routine (ISR) also known as the interrupt service routine handler is defined to enable the external interrupts.
  - Let us learn about the important features which are needed to configure external interrupts in STM32 microcontrollers.

# Interrupt Lines (EXTI0-EXTI15)

- The STM32 ARM microcontroller features 23 event sources which are divided into two sections.
- The first section corresponds t external pins on each port which are P0-P15.
- The second section corresponds to RTC, ethernet, USB interrupts.
- Therefore, in the first section, we have 16 lines corresponding to line0 till line15.
- All of these map to a pin number.

# External Interrupt/Event GPIO mapping




- One important thing to note here is that we can not use two pins in one line at the same time.
- For example, out of PA1, PB1, PC1, PD1, PE1, PF1 and PG1 you can only use a single pin out of all these.
- However, you can use PA1 and PA2 at the same time as they are connected with different lines.
- Now each of these lines EXTI0-EXTI15 can be used to trigger an interrupt on different modes of the signal : rising edge, falling edge or rising_falling edge.

# Interrupt Handler/ Interrupt Service Routine

- The next important feature is the ISR or the interrupt handler. In order to handle the interrupts we use the ISR function.
- The table below shows the interrupt handlers for the GPIO pins 4-0:

| GPIO Pin | IRQ | Handler | Description |
|---|---|---|---|
| 0 | EXTI0_IRQn | void EXTI0_IRQHandler() | Handler for pins connected to line 0 |
| 1 | EXTI1_IRQn | void EXTI1_IRQHandler() | Handler for pins connected to line 1 |
| 2 | EXTI2_IRQn | void EXTI2_IRQHandler() | Handler for pins connected to line 2 |
| 3 | EXTI3_IRQn | void EXTI3_IRQHandler() | Handler for pins connected to line 3 |
| 4 | EXTI4_IRQn | void EXTI4_IRQHandler() | Handler for pins connected to line 4 |

# Interrupt Handler/ Interrupt Service Routine (contd.)

- The table below shows the interrupt handlers for the GPIO pins 9-5:

| GPIO Pin | IRQ | Handler | Description |
|---|---|---|---|
| 5 | EXTI9_5_IRQn | void EXTI9_5_IRQHandler() | Handler for pins connected to line 5 |
| 6 | EXTI9_5_IRQn | void EXTI9_5_IRQHandler() | Handler for pins connected to line 6 |
| 7 | EXTI9_5_IRQn | void EXTI9_5_IRQHandler() | Handler for pins connected to line 7 |
| 8 | EXTI9_5_IRQn | void EXTI9_5_IRQHandler() | Handler for pins connected to line 8 |
| 9 | EXTI9_5_IRQn | void EXTI9_5_IRQHandler() | Handler for pins connected to line 9 |

# Interrupt Handler/ Interrupt Service Routine (contd.)

- The table below shows the interrupt handlers for the GPIO pins 15-10:

| GPIO Pin | IRQ | Handler | Description |
|---|---|---|---|
| 10 | EXTI15_10_IRQn | void EXTI15_10_IRQHandler() | Handler for pins connected to line 10 |
| 11 | EXTI15_10_IRQn | void EXTI15_10_IRQHandler() | Handler for pins connected to line 11 |
| 12 | EXTI15_10_IRQn | void EXTI15_10_IRQHandler() | Handler for pins connected to line 12 |
| 13 | EXTI15_10_IRQn | void EXTI15_10_IRQHandler() | Handler for pins connected to line 13 |
| 14 | EXTI15_10_IRQn | void EXTI15_10_IRQHandler() | Handler for pins connected to line 14 |
| 15 | EXTI15_10_IRQn | void EXTI15_10_IRQHandler() | Handler for pins connected to line 15 |

## Total number of External Interrupts that may be considered at a time

- We have a total of seven interrupt handlers according to the pin which we will set up as the external interrupt pin.
- This is because Line5-Line9 and Line10-Line15 have the same interrupt handler.
- The IRQ has to be set for (Nested Vector Interrupt Control) NVIC and the handler shows the prototype of the handler function

## External Interrupt using a Push Button

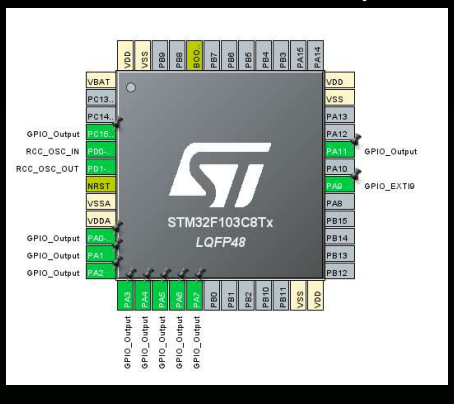
- 8 numbers of LED connected at the 8 pins of Port A will be glowing one after another at every 200 mS from right to left.
- Now we will learn how to handle interrupts in the Blue Pill STM32 using a push button to toggle an LED leaving the movement of glowing of the LED within the previous 8-LEDs.
- The push button will be connected to an interrupt pin of STM32 and configured as an input.
- Whereas the new LED will be set up as a digital output and will be connected to some other pin.
- Upon pressing the push button, in the rising edge of the pulse, the later LED will toggle ten times.
- After that the program will come back to the previous task of the movement of glowing LED.

# Program External Interrupts with STM32 Blue Pill in STM32Cube IDE

- We will use STM32Cube IDE to program our STM32F103C8T6 in the Blue Pill board.
- Open the IDE and head over to a new project

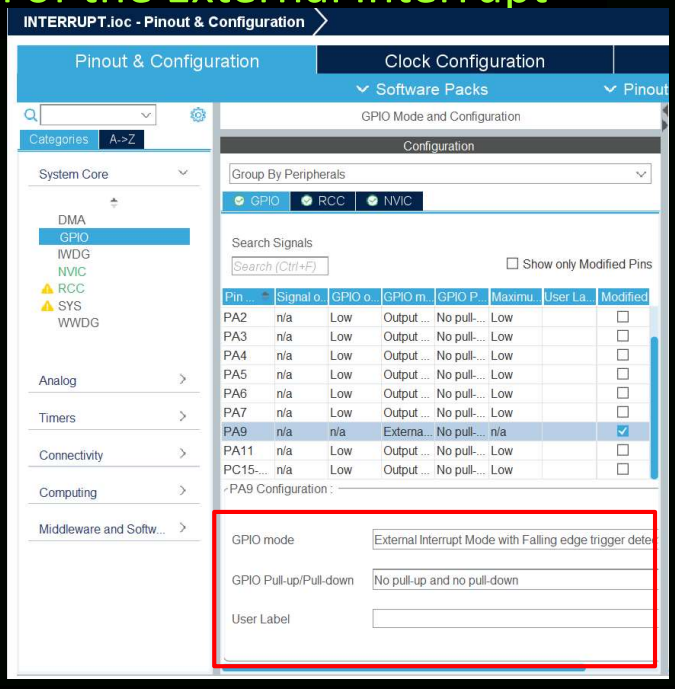
# Device Configuration Tool

- The Device Configuration Tool will now open.
- Here we will configure the external interrupt input pin and the output pins.
- For the output pins we have chosen the digital pins **PA0-PA7 and PA11**.
- Choose **PA9** pin as GPIO_EXTI9 as shown below.
- For the convenience of the connection, PC15 is constantly made zero. So, PC15 is also defined as output.



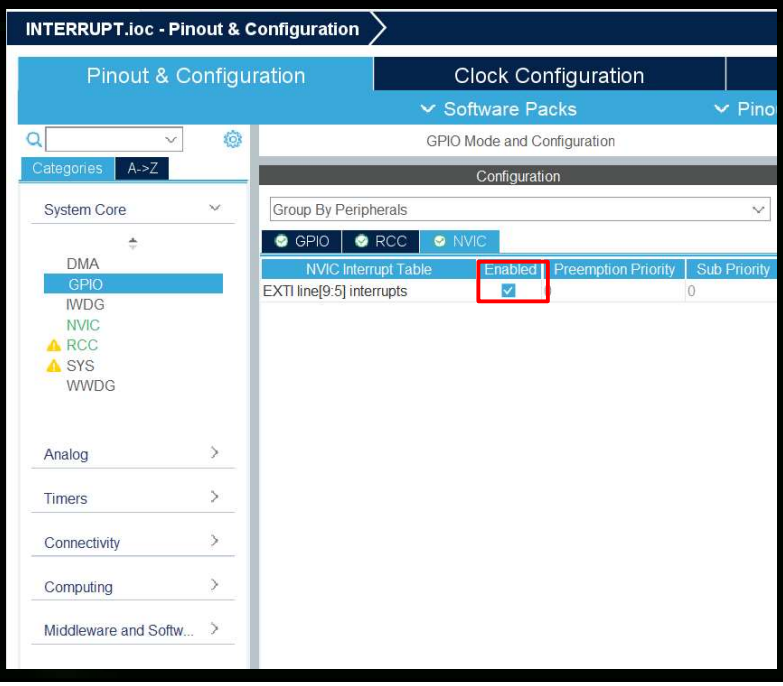
# Changing “GPIO” Configuration of the External Interrupt

- Now go to **System Core > GPIO** and click PA9. Below you will be able to view its configuration.
- Choose “External Interrupt mode with Falling Edge Trigger…”.



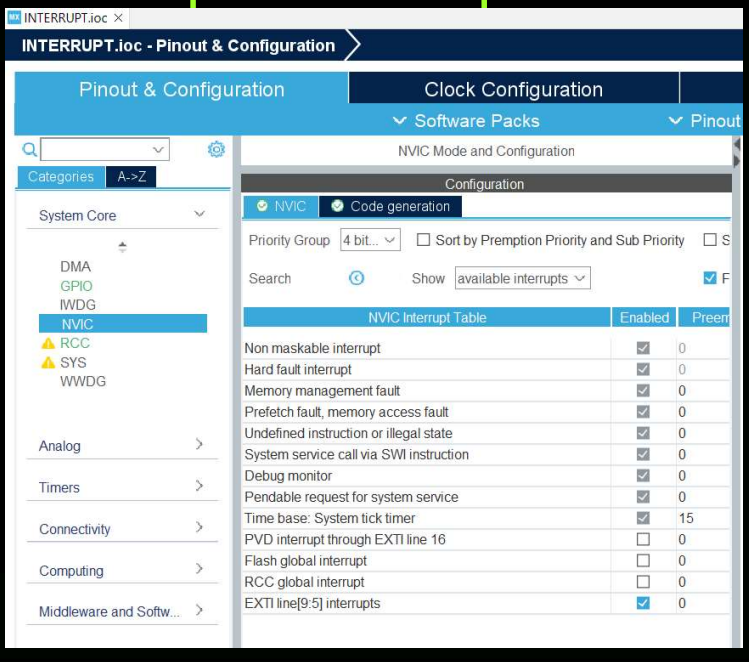

# Changing “NVIC” Configuration of the External Interrupt

- Now click **NVIC** and enable the interrupt for EXTI line 9 as shown.



# Changing Priority of multiple interrupts

- You even have the option to set the priority of the NVIC interrupt. Head over to **System Core > NVIC.**
- Here you will be able to set the priority for EXTI line 9 interrupt.
- Since we are dealing with a single interrupt so, we have left it as default.

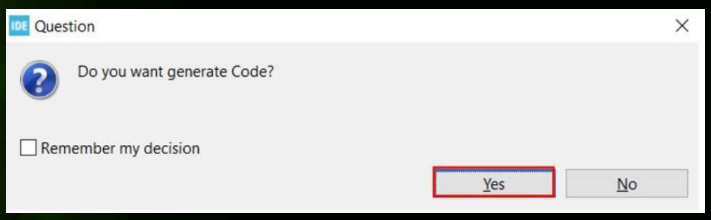



# Setting frequency

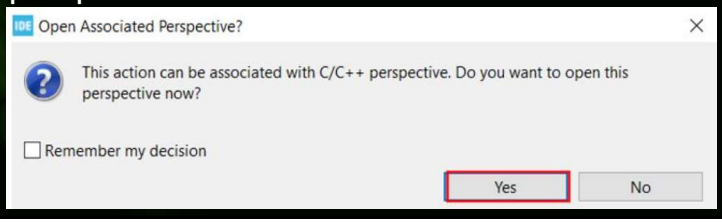
- Now go to **System Core > RCC** then select 'Crystal/Ceramic Resonator' in from the High Speed Clock feature.
- Next go to the Clock Configuration found at the top.
- This will open the following window. Here we will select the clock frequency.
- You can specify your system clock. We will set it as 72 MHz.

# Generating the code

- Now we will save our file. Press Ctrl + S. The following window will appear. Click 'Yes.' This will generate a template code for you.




- Another window will appear that will ask if you want to open the perspective. Click 'Yes'.




# Configuration of the Interrupt Pin

- Now let us look at our main.c file that was generated.
- Look for the MX_GPIO_Init() function
- Here as you can see Pin **A9** is configured as an external interrupt rising edge input with no pull.
- This means that the pin **A9** is setup in high impedance

```
193   /*Configure GPIO pin : PA9 */
194   GPIO_InitStruct.Pin = GPIO_PIN_9;
195   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
196   GPIO_InitStruct.Pull = GPIO_NOPULL;
197   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

# External Interrupt

- The external interrupt is initialized in the following lines within the MX_GPIO_Init() function.

```
199   /* EXTI interrupt init*/
200   HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
201   HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
```

- Here we are first setting the priority of the **EXTI line 9** interrupt and then enabling the IRQ.
- We had already set the mode of the interrupt to rising edge trigger while configuring the input pin.

# Interrupt Handler

- The interrupt channel is enabled. After that let us look at how the handler is implemented.
- You can view the ISR generated in the **stm32f1xx_it.c** file.

```
void EXTI9_5_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI9_5_IRQn 0 */

  /* USER CODE END EXTI9_5_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_9);
  /* USER CODE BEGIN EXTI9_5_IRQn 1 */

  /* USER CODE END EXTI9_5_IRQn 1 */
}
```

# For a Group of External Interrupts

- The void EXTI9_5_IRQHandler(void) function is the ISR that gets called whenever the external interrupt is triggered.
- As we saw in the table previously, EXTI9_5_IRQHandler handles interrupts from more than one pin.
- In our case we are using a single external interrupt pin 9 therefore we called the HAL_GPIO_EXTI_IRQHandler() with the GPIO pin as the parameter inside it once.
- If we had multiple external interrupt pins enabled e.g. pin 7 or pin 8 then we would have called each GPIO handler inside this function.
- Remember: For pin 0 to 4 interrupts, there is only one GPIO handler to call.

# The Callback Function Call

```
void EXTI9_5_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI9_5_IRQn 0 */

  /* USER CODE END EXTI9_5_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_9);
}

/**
  * @brief  This function handles EXTI interrupt request.
  * @param  GPIO_Pin: Specifies the pins connected EXTI line
  * @retval None
  */
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if (__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != 0x00u)
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Callback(GPIO_Pin);
```

Press 'F2' for focus

# Modifying the Code

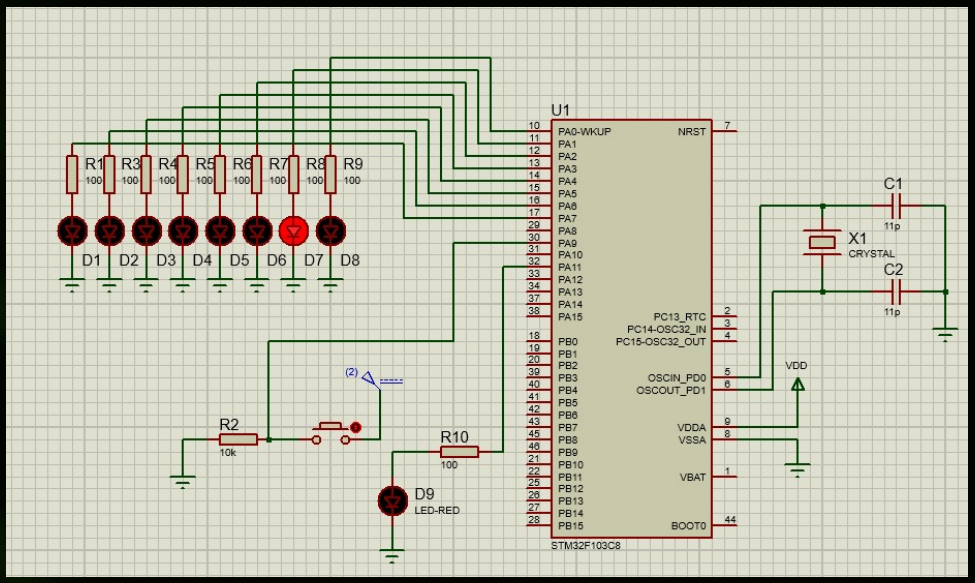
- Inside the main.c file, after the main() function insert this **HAL_GPIO_EXTI_Callback()** function.
- This is the external interrupt ISR handler callback function which is responsible to check the interrupt pin source, then toggle the output GPIO pin accordingly.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
   if(GPIO_Pin == GPIO_PIN_9)
   {
        for (int m=0; m<5; m++) {
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, 1);
              mDelay();
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_11, 0);
              mDelay();
        }
   }
}
```

- One thing to notice here is that only the pin number is being accessed by the callback function.
- Therefore we can configure interrupts on all 16 different pins regardless of the GPIO number.

```
void mDelay(void)
{
   for (int j=0; j<2000; j++){
      for(int k=0; k<1000; k++){
      }
   }
}
```

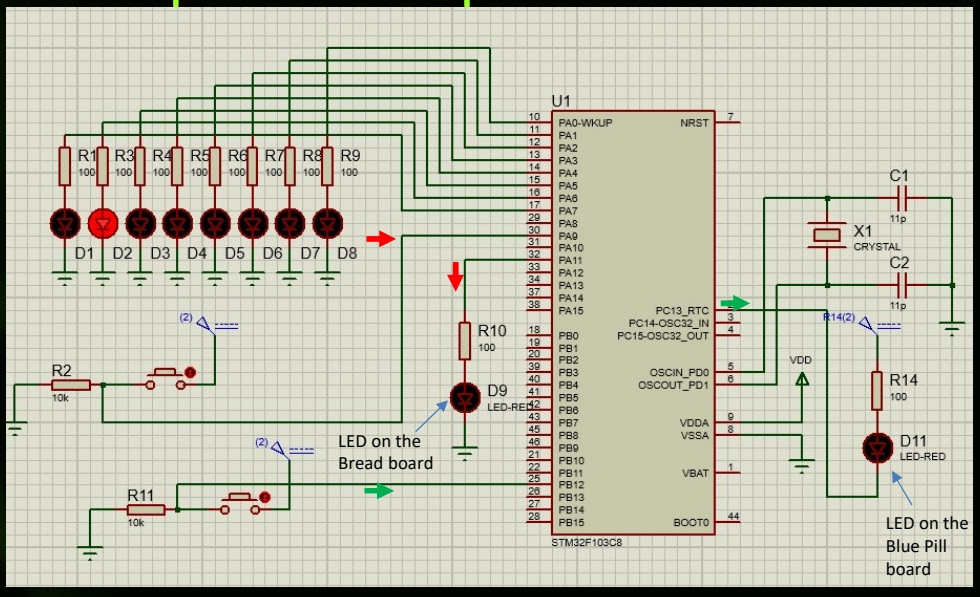
# Proteus Simulation



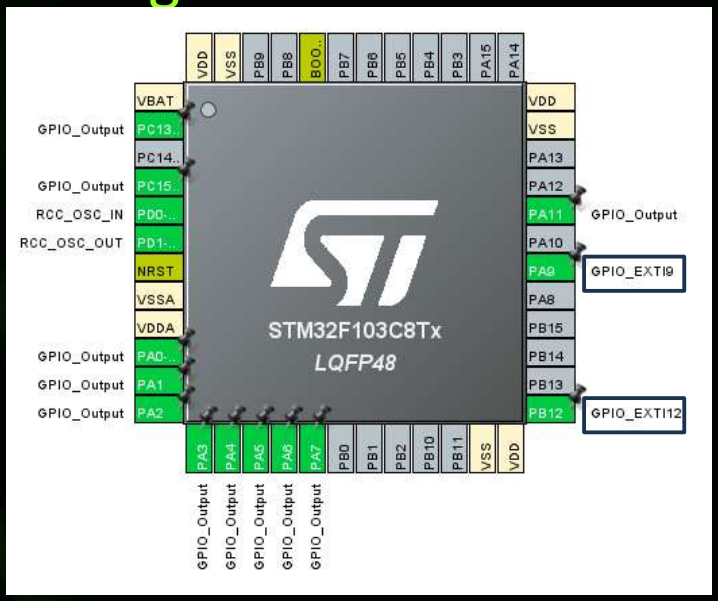
# What is effect of the Priority Setting in case of multiple interrupts?

- When a main task is interrupted by an interrupt, the main task goes in pending mode, the interrupted task start executing.
- But, while the interrupted task while executing a second interrupt will occur, whether the former one will be interrupted or the later one will start executing?
- Find out the answer from the demonstration.

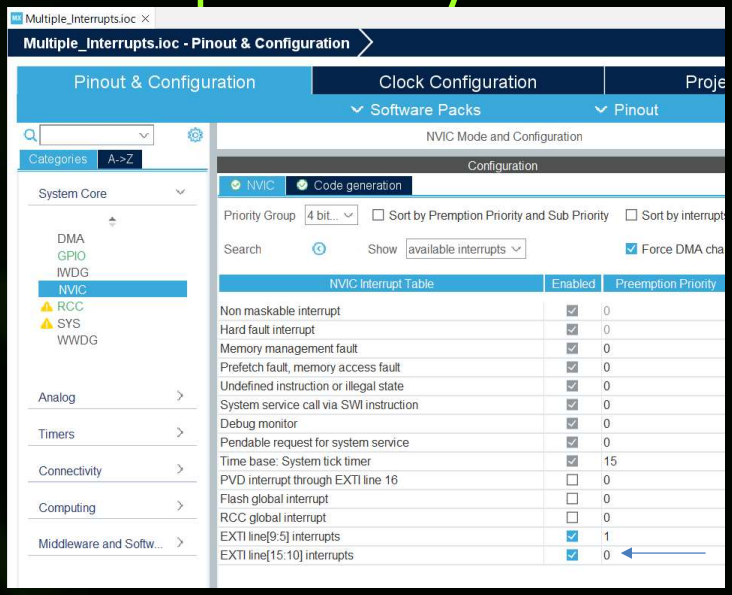
# Multiple interrupt



# Configuration




# Interrupt Priority




Higher

## Assignment

- Develop a project and implement (at least) in Proteus where,
  - Two more interrupts (other that those shown in the class) have to added.
    - One is for Buzzer (Alarm)
    - Other is for Relay Switching.

# Thanks