***EXPERIMENT 4: INTERRUPTS***

***Objectives***

The objectives of Experiment 4 are

- to learn how to use interrupt peripherals

***Apparatus Required:***

- STM32CubeMx
- Keil µVision (MDK ARM)
- STM32 ST-Link Utility
- STM32F4 Microcontroller
- A Jumper Cable

***Preliminary Work:***

1. Study the Interrupt (L05) notes
2. Write the codes of the experimental work in Keil µVision.

***Experimental Work:***

1. Create a new project in CubeMx. Select STMF407VGTx and then STM32F407G-DISC1. First adjust the Pinout&Configuration settings. Close the unnecessary pins. Select the PA0 pin as GPIO_EXTI0 and PA1 pin as GPIO_EXTI1. Select the PD 12-13-14-15 pins as GPIO_Output.

2. Come to the System Core menu. You can change the pin configurations by selecting related pins from here (Figure 1). Select the pull down for PA0&PA1 pins. Select “Output Push Pull” for the GPIO Mode, “Low” for the GPIO output level & Maximum output speed for related pins (PD12- PD13- PD14- PD15).

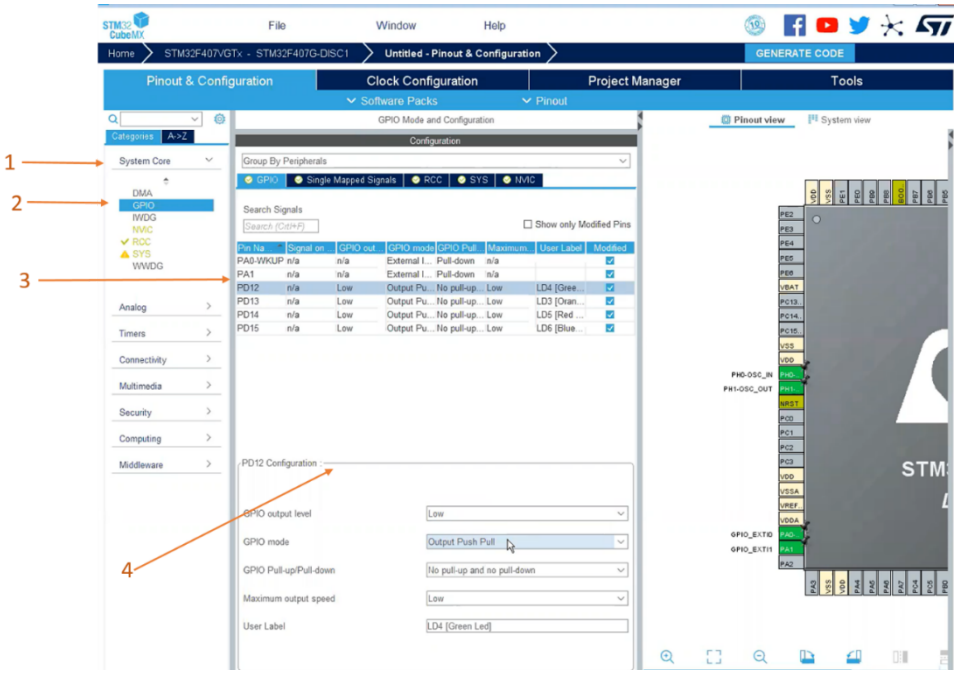



Figure 1

3. Come to the NVIC menu. Firstly set the enable mode for the EXTI line0 & EXTI line1( Figure 2, 1 and 2 steps ). Then, identify the priority levels of the interrupts. Select the Priority Group as 2 bits (which indicate how many bits are needed to identify the priority level) (Figure 2, 3. step). Then, select preemption priority as 1 for the EXTI0 and as 2 for the EXTI1 (Figure 2, 4. step).

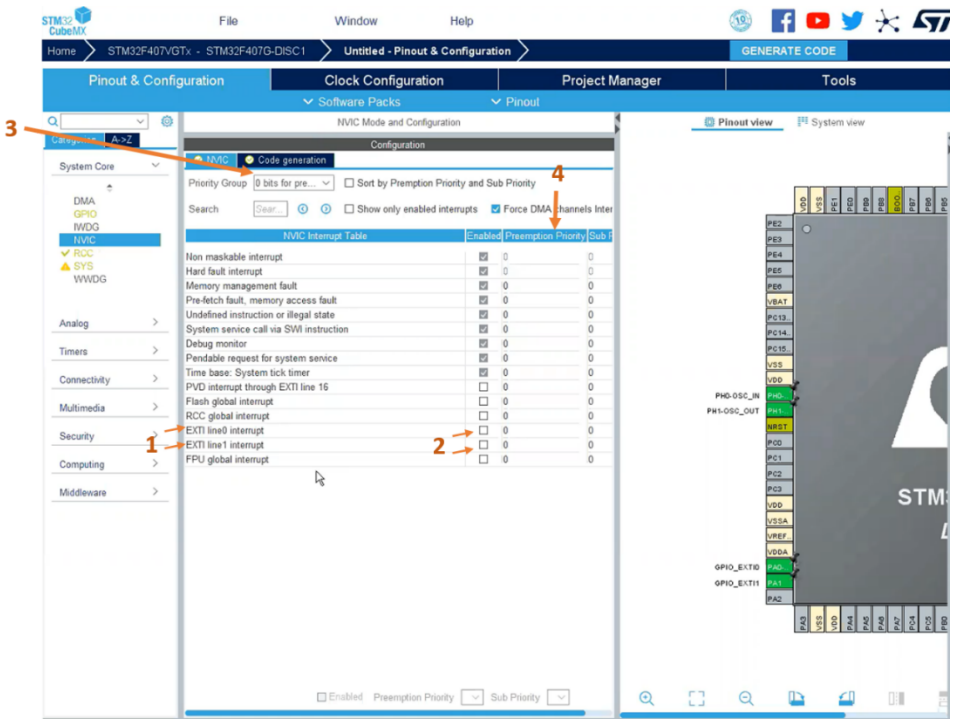



Figure 2

4. Come to the Clock Configuration menu and control the settings as in Figure 3.

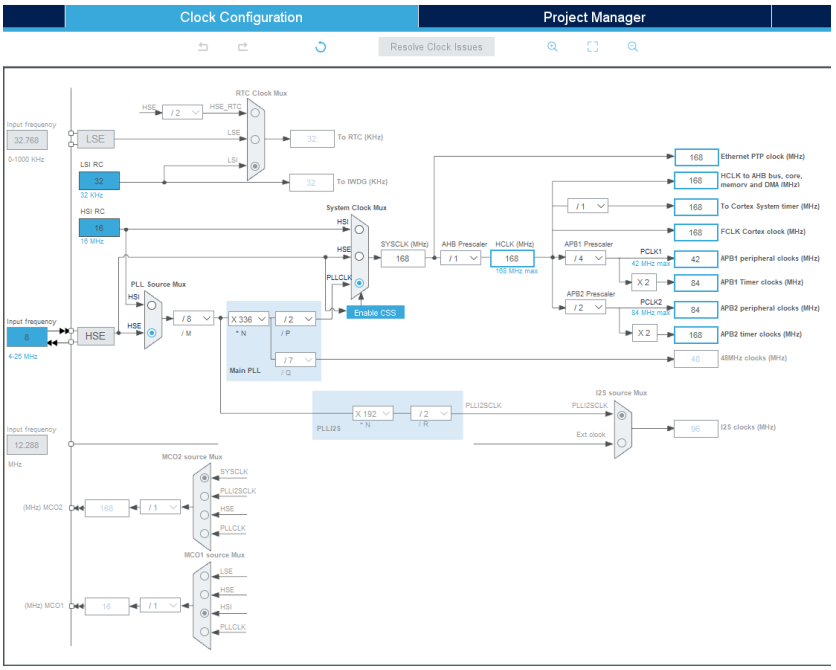



Figure 3

5. Come to the Project Manager and adjust necessary settings as in Figure 4. Then click the Generate Code (Figure 4). Keil µVision programme will open. We can see the settings which were done by CubeMx in the main.c file in Keil µVision (Figure 5). You can change the adjustments from here without going back to the CubeMx. Build the codes in the main.c file. Double click the interrupt file (stm32f4xx.c). We can see the interrupts functions here (Figure 6). We can write codes in the functions. If we want to understand what the function does, we can right click on the function and select the 'Go to Definition…' shown as in Figure 7.

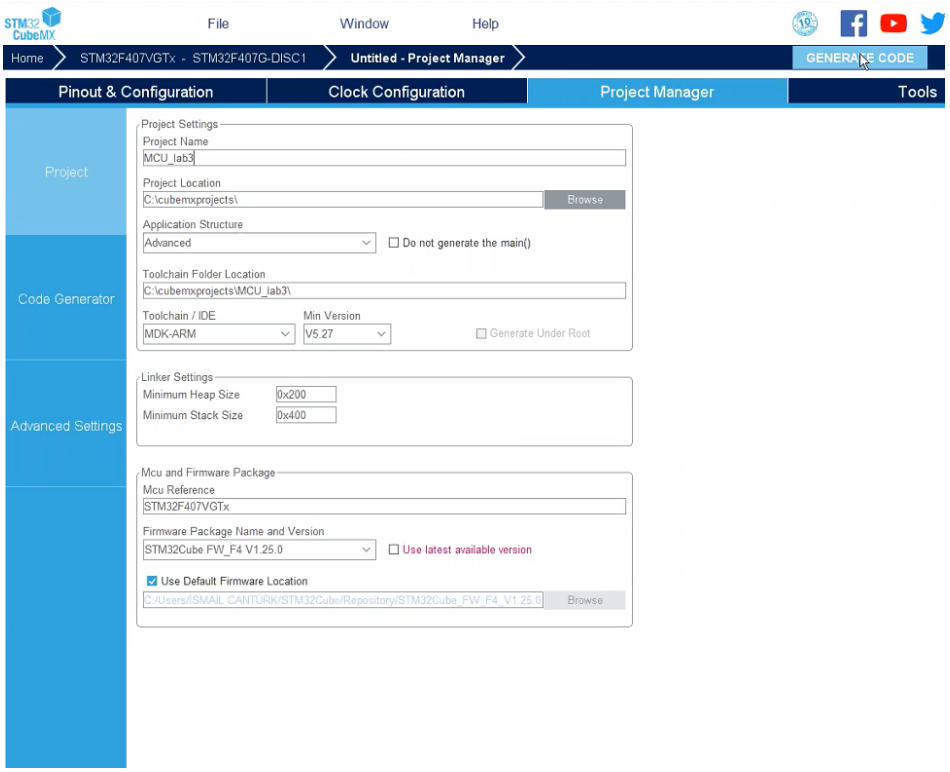



Figure 4

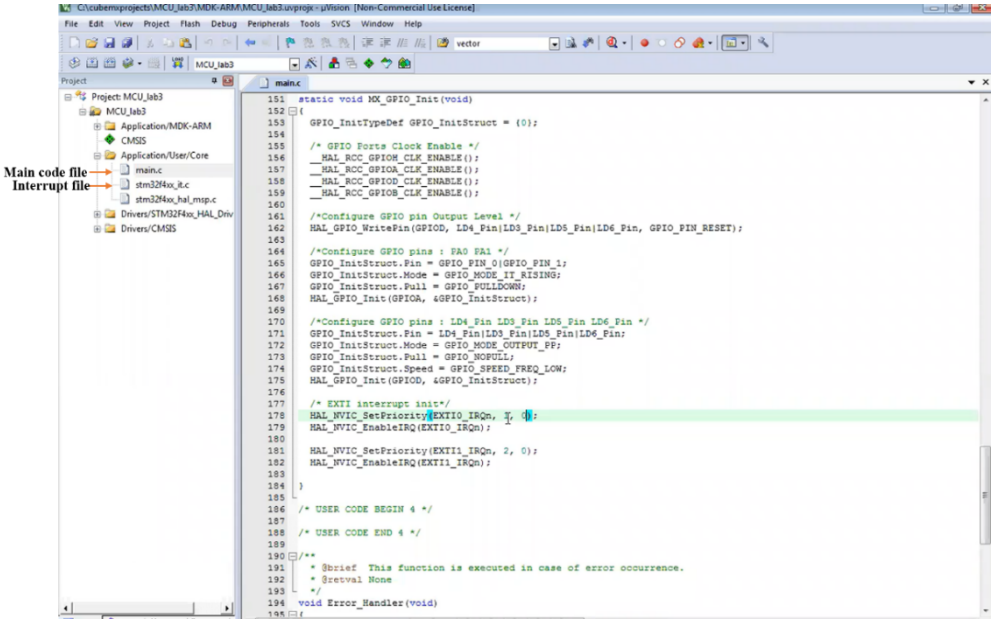
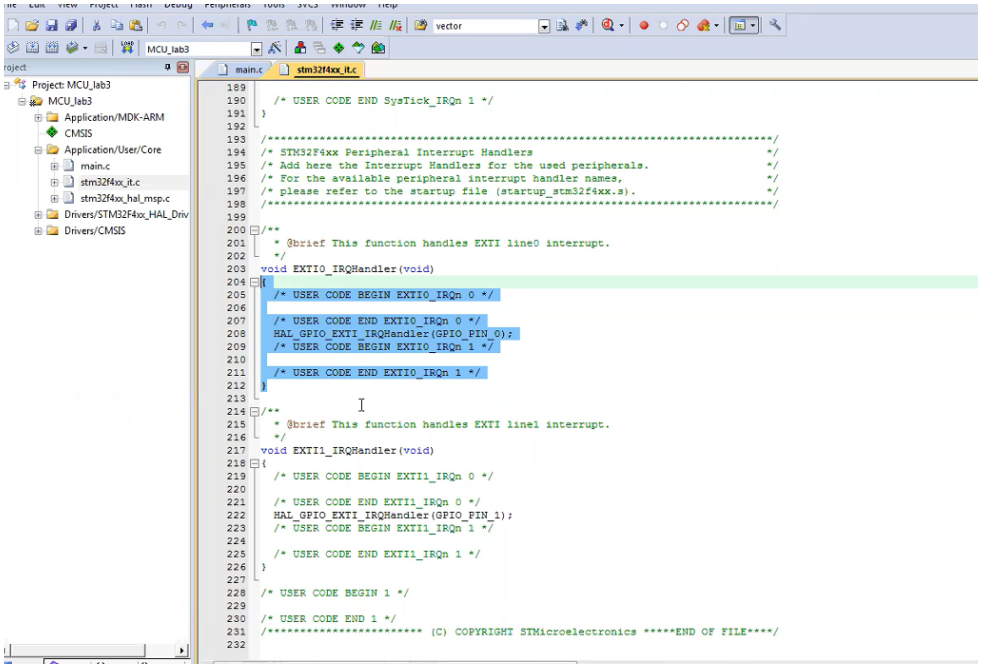
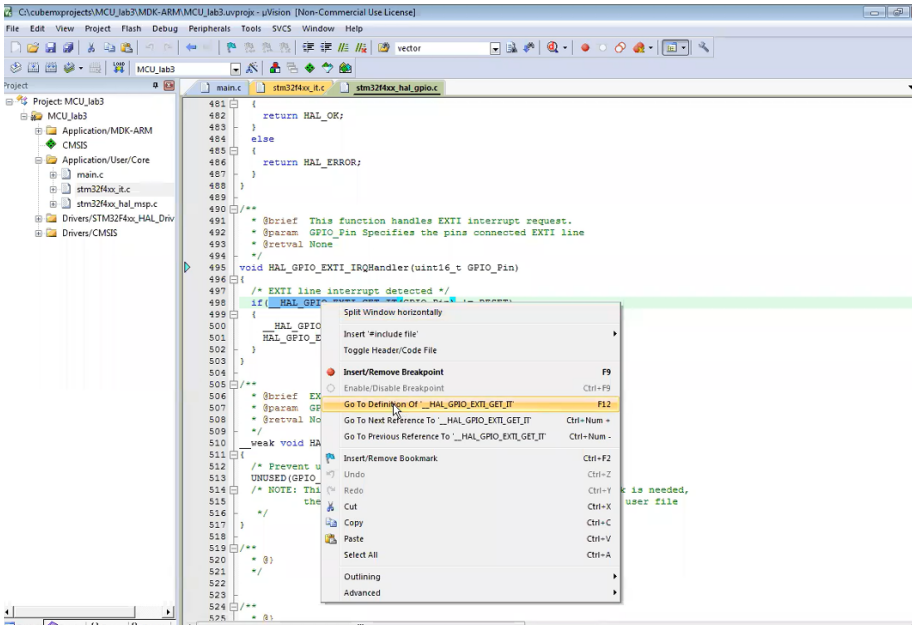



Figure 5



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

/******************************************************************************/
/* STM32F4xx Peripheral Interrupt Handlers                                    */
/* Add here the Interrupt Handlers for the used peripherals.                  */
/* For the available peripheral interrupt handler names,                      */
/* please refer to the startup file (startup_stm32f4xx.s).                    */
/******************************************************************************/
/**
  * @brief This function handles EXTI line0 interrupt.
  */
void EXTI0_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI0_IRQn 0 */

  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */

  /* USER CODE END EXTI0_IRQn 1 */
}

/**
  * @brief This function handles EXTI line1 interrupt.
  */
void EXTI1_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI1_IRQn 0 */

  /* USER CODE END EXTI1_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
  /* USER CODE BEGIN EXTI1_IRQn 1 */

  /* USER CODE END EXTI1_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Figure 6



```
  {
    return HAL_OK;
  }
  else
  {
    return HAL_ERROR;
  }
}

/**
  * @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);
  }
}

/**
  * @brief  EXTI line detection callbacks
  * @param  GPIO_Pin Specifies the pins connected EXTI line
  * @retval None
  */
__weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function should not be modified, when the callback is needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the user file
   */
}

/**
  * @}
  */

/**
```

Figure 7

6. **a.** Write an interrupt handler function that increases by 1 the value of the variable i if an interrupt is generated from the PA0 pin. Write the codes in EXTI0_IRQHandler function in stm32f4xx_it.c file.
   **b.** Write an interrupt handler function that increases by 1 the value of the variable a if an interrupt is generated from the PA1 pin. Write the codes in EXTI1_IRQHandler function in stm32f4xx_it.c file.
   Don't forget to identify variable a and i variables in private variables part of the stm32f4xx_it.c file.
   Observe the change of the i variable when we push the button using debug.

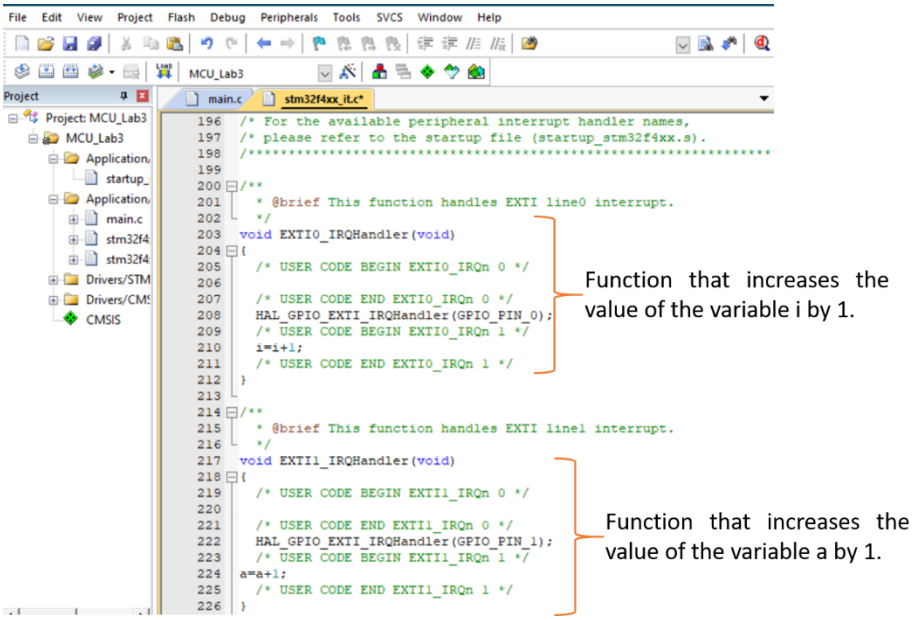



Figure 8

7. Follow the instructions given in a, b, c in order. The relevant codes are given below.
   **a.** When there is no interrupt, the LED connected to the 12th pin lights up continuously. (Write the relevant code inside the while loop in main.c).

```
while (1)
{
 /* USER CODE BEGIN 3 */
 //Light the 12th pin when the interrupt handler is not working
 HAL_GPIO_TogglePin(GPIOD,GPIO_PIN_12); // Toggle the PD12 LED
 HAL_Delay(100); //Wait 100 ms
}
/* USER CODE END 3 */
```

   **b.** When the interrupt is received from the PA0 pin, the value of the i variable increases by 1. Reset all pins connected to port D using BSRR. After the LED is connected to the PD13 pin lights for 5 seconds, all the pins connected to the D port are reset again. Write the relevant code inside the EXTI0_IRQHandler function.

```
void EXTI0_IRQHandler(void)
{
 HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
 /* USER CODE BEGIN EXTI0_IRQn 1 */
i=i+1; //increase i value by 1
GPIOD->BSRR=0xFFFF0000; //Reset the PD pins
GPIOD->BSRR=0x2000; // Set 1 PD13
HAL_Delay(5000); //Wait 5 s
GPIOD->BSRR=0xFFFF0000; // Reset the PD pins
 /* USER CODE END EXTI0_IRQn 1 */
}
```

**c.** When the interrupt is received from the PA0 pin, the value of the i variable increases by 1. Reset all pins connected to port D using BSRR. After the LED is connected to the PD13 pin lights for 5 seconds, all the pins connected to the D port are reset again. Write the relevant code inside the EXTI1_IRQHandler function.

```
void EXTI1_IRQHandler(void)
{
 HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
 /* USER CODE BEGIN EXTI1_IRQn 1 */
a=a+1; //increase a value by 1
GPIOD->BSRR=0xFFFF0000; //Reset the PD pins
GPIOD->BSRR=0x4000; // Set 1 PD14
HAL_Delay(5000); //Wait 5 s
GPIOD->BSRR=0xFFFF0000; // Reset the PD pins
 /* USER CODE END EXTI1_IRQn 1 */
 }
```

**d.** Compile the codes and upload them to the microcontroller. Observe the change of i and a variable using debug. Use the button on the microcontroller to send an interrupt from the PA0 pin. Use the 5V on the microcontroller discovery card to send the interrupt from the PA pin (You can connect 5V to the PA1 pin with the help of a jumper).

8. Use priorities of the interrupts (Go back to the 3 to remember the priorities of the interrupts). Use the same codes as in 7.
   **a.** After giving an interrupt from PA0 pin, give another interrupt from PA1 pin before the interrupt handler is completed. Observe the changes of i, variables and LEDs. Observe the 'Tail Chaining'.
   **b.** After giving an interrupt from PA1 pin, give another interrupt from PA0 pin before the interrupt handler is completed (Late Arrival). Observe the changes of i, variables and LEDs.

9. Learn how to use the interrupt mask register (Examine the properties of the register from Reference Manual). Write a code inside the EXTI1_IRQHandler function. When a value is greater than 5, mask pin 1 using the Interrupt Mask Register. Use EXTI->IMR statement to reach the interrupt mask register and assign a hexadecimal number to this register that will set the corresponding pin value to zero. Build and Load the code. Use debug to observe the chaining of a value. Write down your observations about what the end result was.

```
void EXTI1_IRQHandler(void)
{
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
  /* USER CODE BEGIN EXTI1_IRQn 1 */
a=a+1;
GPIOD->BSRR=0xFFFF0000;//Reset the PD pins
GPIOD->BSRR=0x4000;// Set 1 PD14
HAL_Delay(5000); //Wait 5 s
GPIOD->BSRR=0xFFFF0000;// Reset the PD  pins
//Since a>5, interrupts from line 1 are not detected
if (a>5)
{
EXTI->IMR=0x7FFFFD; //Masked the 1. pin
}
  /* USER CODE END EXTI1_IRQn 1 */
}
```