Microprocessors and Peripherals
2nd Lab.

Interrupts and ISRs on the STM32 nulceo microcontroller

Authors: Konstantinidis Paschalis, Tzouvaras Evangelos.
Team : 5

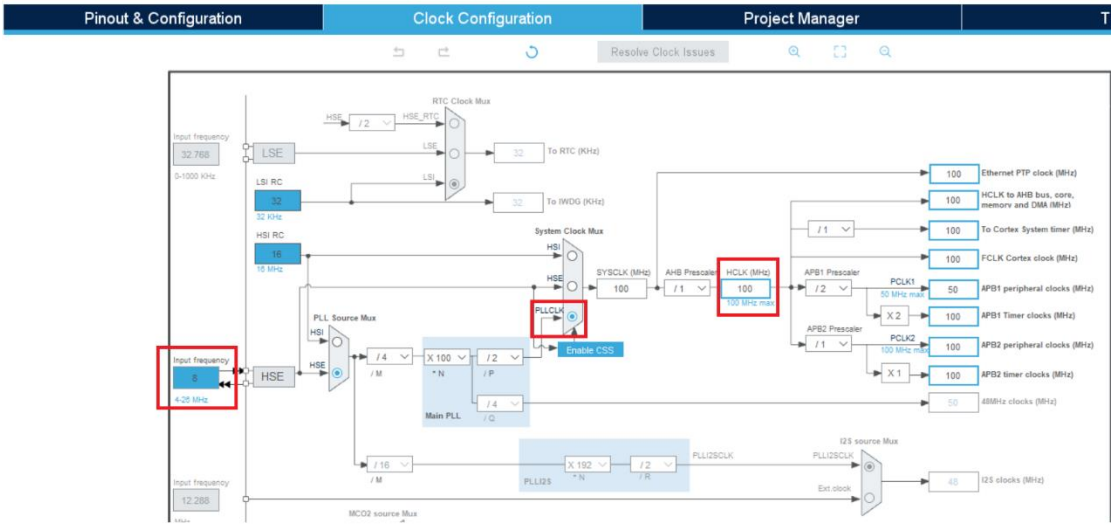
Contents

Lab Description:

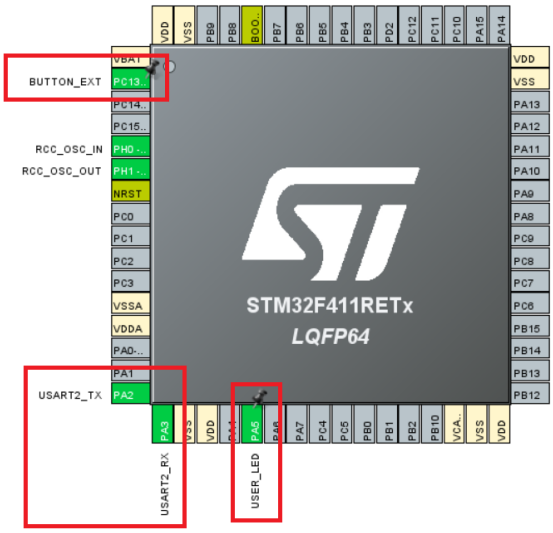
The second lab contains the interrupts and ISR programming technics of the STM32 microcontroller and the connectivity via the UART protocol. The program requests from the user his AEM or to press the external button. On the first case, the program takes the AEM number via UART and the receive interrupt calls the ISR callback. On the second case, if the button was pressed, the program calls the external interrupt ISR callback. Then the program is responsible to transmit via UART appropriate messages to the user. The program was developed by using the STM32 Cube MX and the HAL drivers with cooperation of keil uVision, because we are more familiar with this packet.

STM32 Cube MX Initialization:

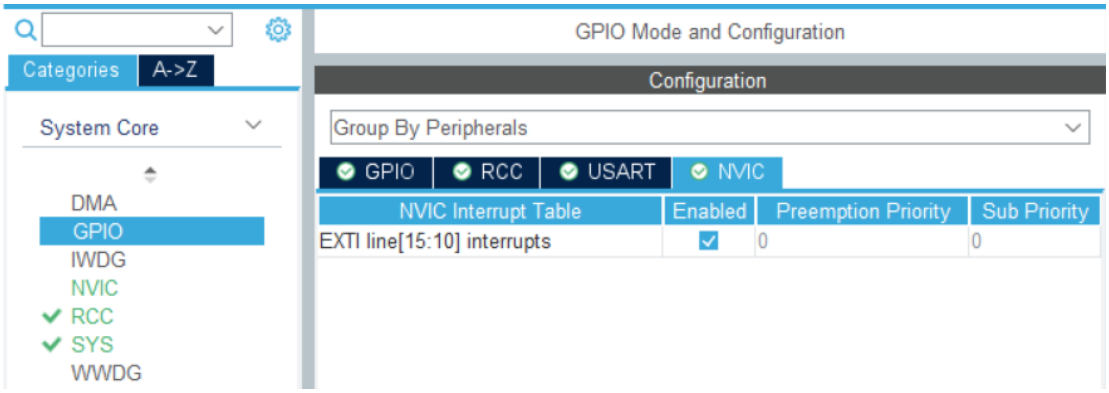
The STM32 Cube MX IDE, is used to initialize the clock and all the peripherals that will be used on the program. First of all, from the clock configuration tab, we set up the clock input frequency at 8MHz and the maximum processor frequency at the 100MHz, as shown in the picture below.



We will use the user LED and the user button that are integrated on the STM32 nucleo board. From the datasheet we can see that the LED is connected on the PA5 pin and the user button is connected to the PC13 pin. From the pinout configuration on the STM32 cube MX, we initialize the PA5 pin to be GPIO_Output, and the PC13 to be GPIO_EXTI_13 (external interrupt). Also we choose the PA2 and PA3 pins to use them for the UART communication as Tx and Rx pins.



The final set up on the Cube MX is to enable the external interrupts. The Cube MX IDE will create all the code on the keil which will initialize all these appropriate peripherals.



UART Initialization:

The UART protocol when will receive every message, it will cause an interrupt and the ISR *HAL_UART_RxCpltCallback* will run. Also, the microcontroller will send on the UART channel messages but using the *HAL_UART_Transmit_IT* function. First of all, we will transmit a message to the user to give his AEM or to press the button.

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
/* USER CODE BEGIN 2 */
// UART initializations
HAL_UART_Receive_IT(&huart2, uart_data, 5);                // Initialize the first interrupt of the UART receive message

// Send via UART a message to the user
HAL_UART_Transmit_IT(&huart2, (uint8_t *)"\nGive me your AEM or press a button\n", strlen("\nGive me your AEM or press a button\n"));

/* USER CODE END 2 */
```

AEM ISR:

If the user gives the AEM via the UART an interrupt will call the *HAL_UART_RxCpltCallback* function. The AEM is saved on the *uint8_t uart_data* array. This array has a length of five digits because we consider that the AEM is four digit number and the final position of the array is used for the enter special character.

On this function, we check if the final digit (uart_data[3]) is even or odd number. In case of an even number, the LED will be turned off by using the *HAL_GPIO_WritePin* function. Otherwise, the LED will be turned on. Also, the program checks if the LED changes its situation by checking its previous state with the current state. If a change on its state is detected, the state_changed

flag will become HIGH. This flag will cause a block at the main function to run, which is responsible to transmit via UART protocol that the LED changed its state. Then we reset the flag value on this block to be ready for the next iteration.

```
// ISR callback for UART receiving data
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
 /* The program receives a 4 digit number (The AEM)
    If the last digit is even number, the ISR will close the LED
    If the last digit is odd number the ISR will light up the LED
 */
 if(huart -> Instance == USART2)        // Check if the data that were received, came from the UART channel 2
 {
  if(uart_data[3] % 2 == 0)             // Check if the uart_data[3] (the last digit of the AEM), is even number
  {
   // Detect a change on the LED's state
   if(led_state == 1)                   // Check the previous state of the led (if the led was lighten)
   {
    state_changed = 1;                  // If the LED was open, then detect that a change will become on its state
    led_state = 0;                      // Make the state LOW (LED off)
   }
   HAL_GPIO_WritePin(USER_LED_GPIO_Port, USER_LED_Pin, GPIO_PIN_RESET);     // If the last digit is even number, make the LED pin LOW (Turn off the LED)
  }
  else{                                 // Else the last number will be odd number. Therefore the LED must light up
   // Detect a change on the LED's state
   if(led_state == 0)                   // Check if the LED state before the change was closed
    {
     state_changed = 1;                 // If the LED was closed, then detect that a change will become on its state
     led_state = 1;                     // Make the state HIGH (LED on)
    }
    HAL_GPIO_WritePin(USER_LED_GPIO_Port, USER_LED_Pin, GPIO_PIN_SET);      // If the last digit is odd number, make the LED pin HIGH (Turn onthe LED)
  }
}
```

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  // If the state of the LED was changed from any of the callbacks, print a message for this change via UART
  if(state_changed == 1)
  {
    HAL_UART_Transmit_IT(&huart2, (uint8_t *)"\nLED CHANGED\n", strlen("\nLED CHANGED\n"));
    HAL_Delay(50);
    state_changed = 0;                  // Reset the state_changed flag (To be ready for use on the next iteration)
  }
```

External button ISR:

If the user presses the external button, the system will call the *HAL_GPIO_EXTI_Callback* automatically, because we set the button pin as external interrupt pin on the STM32 Cube MX IDE. Then the program will change the LED state by using the *HAL_GPIO_TogglePin* function. Also the callback enables the state_changed flag and the print_button flag. These flags will cause special instructions to be run on the main function, and the program will send appropriate messages to the user via the UART protocol. More specifically, the state_changed flag will transmit the message to the user that the LED changed its state due to the button press, and the print_button flag will transmit the number of the button press.

```c
// ISR callback for the external interrupt (For the button)
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  // If the user will press the button, then this function will change the LED's state
   HAL_GPIO_TogglePin(USER_LED_GPIO_Port, USER_LED_Pin);
   state_changed = 1;              //  Detect that the LED changed its state
   print_button = 1;               // Make the print_button flag 1, in order to run the appopriate block on the main function
   counter_button += 1;            // Increase the counter, that counts the times that the LED was pressed, by 1
}
/* USER CODE END 4 */
```

```c
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
  // If the state of the LED was changed from any of the callbacks, print a message for this change via UART
  if(state_changed == 1)
  {
    HAL_UART_Transmit_IT(&huart2, (uint8_t *)"\nLED CHANGED\n", strlen("\nLED CHANGED\n"));
    HAL_Delay(50);
    state_changed = 0;          // Reset the state_changed flag (To be ready for use on the next iteration)
  }

  // If the button was pressed, print via UART the times that the button was pressed
  if(print_button == 1)
  {
    HAL_UART_Transmit_IT(&huart2, (uint8_t *)"\nTimes pressed: ", strlen("\nTimes pressed: "));       // Transmit via UART the message about the times that the button was pressed
    HAL_Delay(50);
    sprintf(counter_button_string, "%d", counter_button); // Convert the counter_button integer variable to string in order to be sent via UART
    HAL_UART_Transmit_IT(&huart2, (uint8_t *)counter_button_string, 2);              // Send the counter_button_string via UART
    HAL_Delay(50);
    print_button = 0;           // Reset the print_button flag (To be ready for use on the next iteration)
  }


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

Testing and problems:

For the testing we use the STM32F401 nucleo board, so the button and the LED are integrated on the board. As for the UART console, we used the TeraTerm application to communicate as users with the microcontroller. Two problems are appeared during the testing. The first problem was about the UART messages. Due to the high clock speed of the processor and to the multiple messages on the main, the UART could not transmit the messages correctly. For this reason we placed some delays on the program (*HAL_Delay(50)*), to transmit the messages via UART correctly. The second problem was about the way to transmit an integer number (the count_button variable), via the UART protocol which sends only uint8_t data. To solve this problem we used the *sprintf* function to convert an integer into a two dimension array number to transmit the count_button variable via UART.