GEBZE TECHNICAL UNIVERSITY

ENGINEERING FACULTY

ELECTRONICS ENGINEERING

ELEC 335

MICROPROCESSORS

LAB 03

Name - Surname	Yağmur DERYA
Student ID	171024011

1. Introduction

In this lab, the goal is to understand exception operation, fault generation and handling, external interrupt and managing multiple sources and priority. Except inline assembly, C language is used for the problems.

2. Problems

2.1. Problem 1 – HardFault Handler

In this problem, a HardFault Handler which detects possible faults such as;

- Bus error – an attempt to access an invalid address
- Unaligned memory access
- Undefined instruction execution

is created.

An external button is added to create faults when it is pressed and used the onboard LED to see the reset operation. When the button is pressed, fault happens and the LED is turned off. HardFault Handler restores stack pointer and calls reset handler. Reset handler calls the main function and the program starts again.

```
/* HardFault Handler */
void HardFault_Handler(void){
	GPIOC -> ODR &= (0U << 6); // turn of the LED
	__asm("ldr r0, =_estack");
	__asm("movs sp, r0");
	NVIC_SystemReset(); // resets the system
	__asm("b Reset_Handler");
}
```

```
/* Reset Handler */
void Reset_Handler(void){
	main();
	for(;;);
}
```

```
int main(void) {

    /* Enable GPIOC clock */
    RCC->IOPENR |= (1U << 2);
    /* Enable GPIOB clock */
    RCC->IOPENR |= (1U << 1);
    /* to enable GPIOA clock : (1U << 0) */

    /* Setup PC6 as output */
    GPIOC->MODER &= ~(3U << 2*6); // clear bits ~(11) = 00
    GPIOC->MODER |= (1U << 2*6); // set PC6 as output : 01


    /* Setup PB3 as input */
    GPIOB->MODER &= ~(3U << 2*3); // to set PB3 as input : 00

    int fault = 1;

    while(1) {
      if(fault == 0){
                    if((GPIOB->IDR >> 3) & 1){ // button active
```

```
                        __asm("ldr r6, =0xFFFFFFE8"); // invalid address
bus error
                        __asm("ldr r5, [r6]");
                  } else {
                        /* turn on LED on PC6 */
                        GPIOC->ODR |= (1U << 6);
                  }
      } else if (fault == 1){
            if((GPIOB->IDR >> 3) & 1){
                  __asm("ldr r6, =0x80000101"); // unaligned memory
                  __asm("ldr r5, [r6]"); // read word
            } else {
                  GPIOC->ODR |= (1U << 6);
            }
      } else {
            if((GPIOB->IDR >> 3) & 1){
                  __asm("ldr r6, =0x81000100");
                  __asm("ldr r5, [r6]");
            } else {
                  GPIOC->ODR |= (1U << 6);
            }
      }
    }

    return 0;
}
```

2.2. Problem 2 – External Interrupt Routine

In this problem, onboard LED is used. When the external interrupt happens, which is pushing the button, LED is toggled.

```
    void EXTI2_3_IRQHandler(void){

        GPIOC->ODR ^= (1U << 6); // toggle
        EXTI->RPR1 |= (1U << 3); // clear pending
    }
```

```
    int main(void) {

        /* Enable GPIOC clock */
        RCC->IOPENR |= (1U << 2);
        /* Enable GPIOB clock */
        RCC->IOPENR |= (1U << 1);


        /* Setup PC6 as output */
        GPIOC->MODER &= ~(3U << 2*6);
        GPIOC->MODER |= (1U << 2*6);

        /* Turn on LED */
        GPIOC->ODR |= (1U << 6);


        /* Setup PB3 as input */
        GPIOB->MODER &= ~(3U << 2*3);

        EXTI->RTSR1 |= (1U << 3); // rising trigger selection register
```

```
        EXTI->EXTICR[0] |= (1U << 8*3); // external interrupt selection
    register
        EXTI->IMR1 |= (1U << 3); // wake up with interrupt mask register

        NVIC_SetPriority(EXTI2_3_IRQn, 0);
        NVIC_EnableIRQ(EXTI2_3_IRQn);

        while(1) {
        }
        return 0;
    }
```

2.3. Problem 3 – External Interrupts with Multiple Sources

Two buttons and LEDs are connected to the board and external interrupts are enabled for both buttons. When a button is pressed, a LED is turned off, waits and turned on just before the interrupt finish.

Because buttons have different priorities, when higher priority one works every time, lower priority one waits its turn.

I gave higher priority to the button which is connected to PB3 and toggle the LED which is connected to PB6 with this button.

```
void EXTI2_3_IRQHandler(void){
    GPIOB->ODR &= ~(1U << 6); // turn off LED on PB6
    for(volatile long long int a = 0; a < 5000000; ++a);
    GPIOB->ODR |= (1U << 6);
    EXTI->RPR1 |= (1U << 3); // clear pending
}

void EXTI0_1_IRQHandler(void){
        GPIOB->ODR &= ~(1U << 4); // turn off LED on PB4
        for(volatile long long int a = 0; a < 5000000; ++a);
        GPIOB->ODR |= (1U << 4);
        EXTI->RPR1 |= (1U << 1); // clear pending
}
```

```
int main(void) {

    /* Enable GPIOB clock */
    RCC->IOPENR |= (1U << 1);

    /* Setup PB6 as output */
    GPIOB->MODER &= ~(3U << 2*6);
    GPIOB->MODER |= (1U << 2*6);

    /* Setup PB4 as output */
    GPIOB->MODER &= ~(3U << 2*4);
    GPIOB->MODER |= (1U << 2*4);

    /* Setup PB3 as input */
    GPIOB->MODER &= ~(3U << 2*3);
```

```
    EXTI->RTSR1 |= (1U << 3); // rising trigger selection register
    EXTI->EXTICR[0] |= (1U << 8*3); // external interrupt selection
register
    EXTI->IMR1 |= (1U << 3); // wake up with interrupt mask register

    NVIC_SetPriority(EXTI2_3_IRQn, 0); // higher priority
    NVIC_EnableIRQ(EXTI2_3_IRQn);

    /* Setup PB1 as input */
    GPIOB->MODER &= ~(3U << 2*1);

    EXTI->RTSR1 |= (1U << 1); // rising trigger selection register
    EXTI->EXTICR[0] |= (1U << 8*1); // external interrupt selection
register
    EXTI->IMR1 |= (1U << 1); // wake up with interrupt mask register

    NVIC_SetPriority(EXTI0_1_IRQn, 1); // lower priority
    NVIC_EnableIRQ(EXTI0_1_IRQn);

    while(1) {
        /* Turn on LED on PB6 */
        GPIOB->ODR |= (1U << 6);
        /* Turn on LED on PB4 */
        GPIOB->ODR |= (1U << 4);
    }
    return 0;
}
```

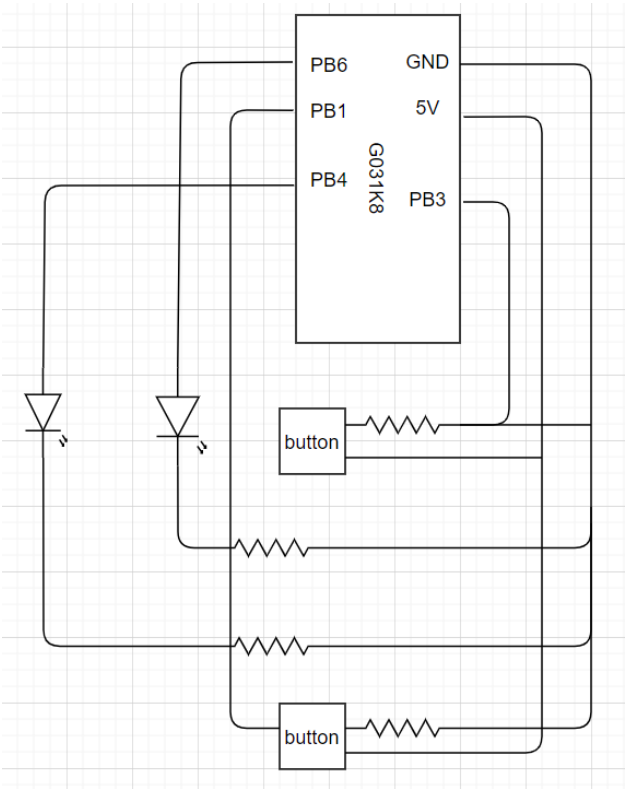

Figure 1. Diagram.

3. References

[1] RM0444 Reference manuel

[2] https://github.com/fcayci/stm32g0

[3] https://www.iar.com/support/tech-notes/debugger/debugging-a-hardfault-on-cortex-m/