GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC335

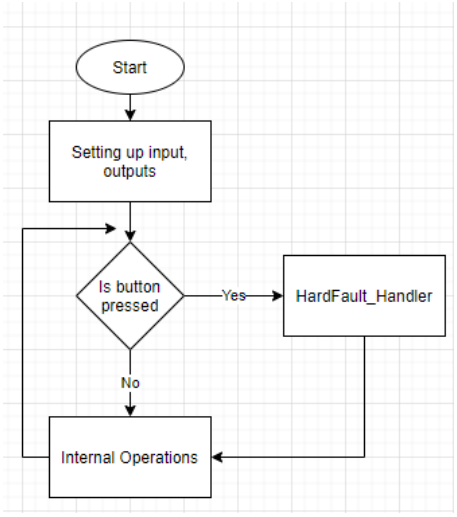
MICROPROCESSORS LAB

LAB 3 Report

Prepared by
171024086 Berat KIZILARMUT

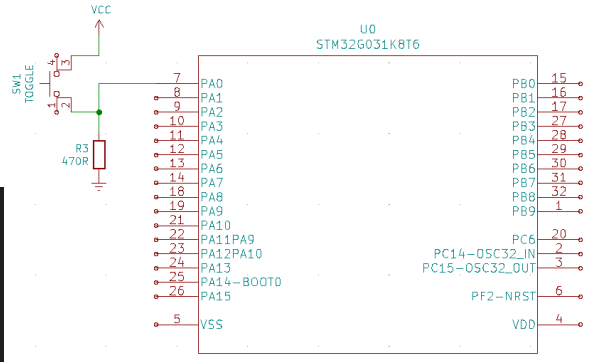
1. Introduction

Aim of this lab is to get a better understanding about exception and interrupt operations. We used manual button check operations to operate a push button on prior homework and labs, now we will begin utilizing interrupt functions. Utilized the \stm32g0-main\blinky project as a template on every problem.

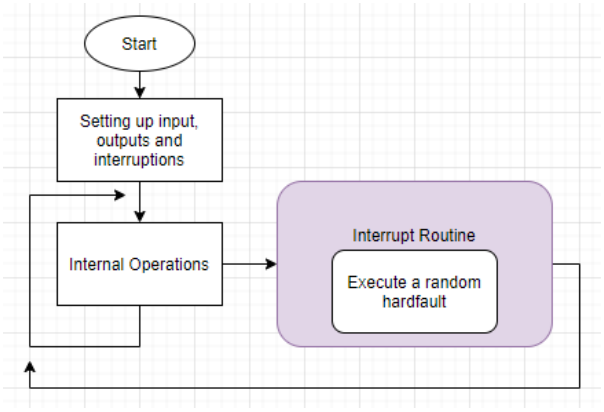


2. Problem 1

In this project we will design our own Hardfault Handler. We will connect a button. In the infinite loop where the internal functions of a given program will be executed. Before the program execution, if statement will check for button input, if there is an button input hardfault handler will be called.



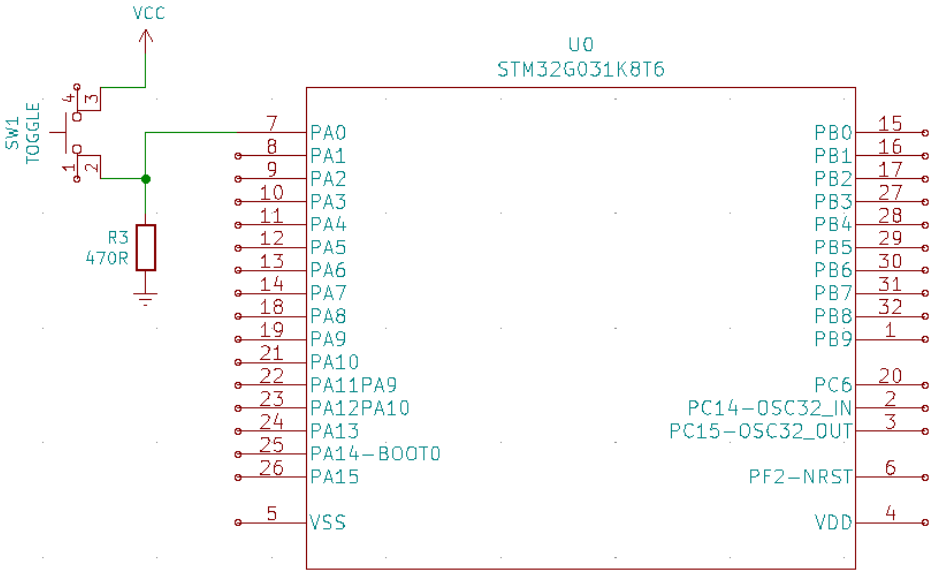

```c
#include "stm32g0xx.h"

void HardFault_Handler(void) {
    /* Insert Hardfault Functions Here */ }
int main(void) {
    /* Setting up the button and interruption */
    /* Enable GPIOA clock */
    RCC->IOPENR |= (1U);
    /* Setup PA0 as input */
    GPIOA->MODER &= ~(3U);
    while(1) {
        if(GPIOA->IDR & 1U) { /* Check for button input */
            HardFault_Handler(); /* If button is one, call hardfault handler */
}
        /*Internal Functions */
    } /* Endless loop */
    return 0;
}
```



3. Problem 2

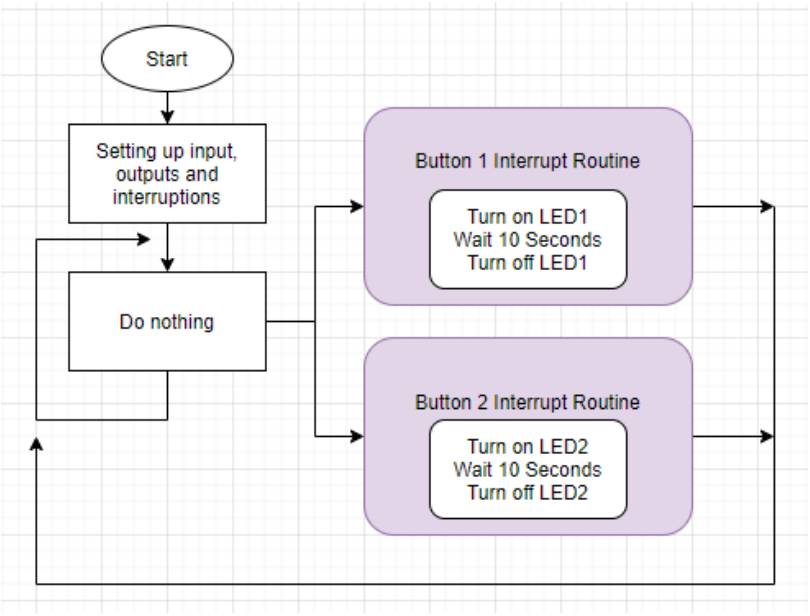
In this problem we will use external interrupt routines for the first time. These routines are much more sophisticated than the manual button check operations I had to use before. This said implemented external interrupt routine will be tied to random hardfaults.

C Code starts with setting the peripheral clocks for GPIOA and setting PA0 for input to connect the push button. After the peripheral setting operations, external interrupt operation is set. RTSR1 is set to trigger on rising edge of the button. EXTICR[0], external interrupt selection register is set without any for PA0. EXTI interrupt mask register, IMR, is set 1 for PA0. Rising edge pending register is set 1 to clear PA0. NVIC priorty set 0 for up most priority. NVIC_EnableIRQ is set EXTI0_1_IRQn to activate EXTI0_1 operations. After this its time to override the default EXTI0_1IRQHandler because it is set as weak by default. We can override it by defining it in the main.c and putting the hardfault functions there.



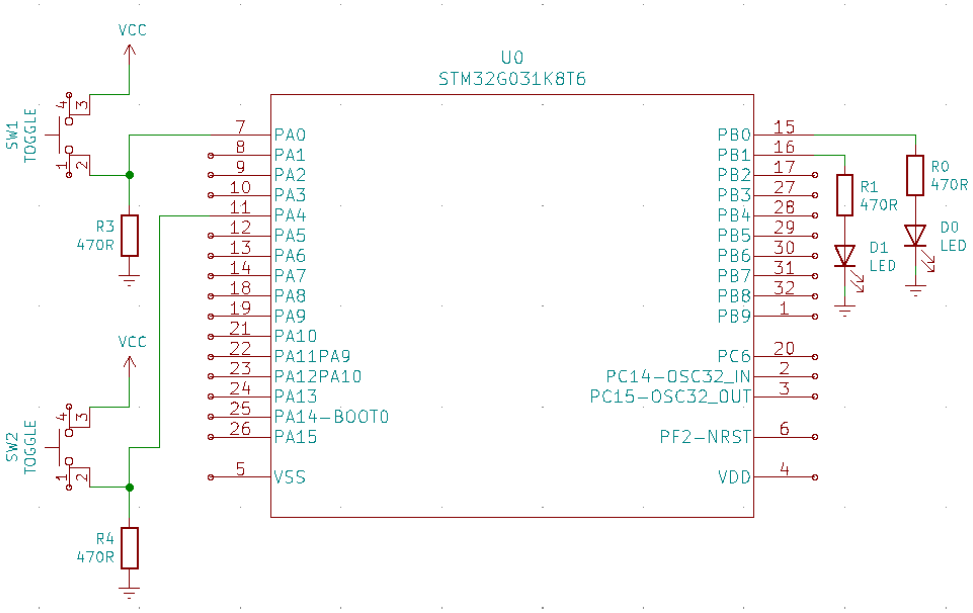

```c
#include "stm32g0xx.h"

void HardFault_Handler(void) {
    /* Insert Hardfault Functions Here */ }

void EXTI0_1IRQHandler (void) {
    HardFault_Handler();
    EXTI->RPR1 |= (1U << 0); }

int main(void) {
    /* Setting up the button and interruption */
    /* Enable GPIOA clock */
    RCC->IOPENR |= (1U);
    /* Setup PA0 as input */
    GPIOA->MODER &= ~(3U);

    /* Setting up the interrupt operation for PA0 */
    EXTI->RTSR1 |= (1U); /* Setting the trigger as rising edge */
    EXTI->EXTICR[0] &= ~(1U << 8*0); /* EXTICR 0 for 0_1 */
    EXTI->IMR1 |= (1U << 0); /* Interrupt mask register */
    EXTI-
>RPR1 |= (1U << 0); /* Rising edge pending register, Clearing pending PA0 */
    NVIC_SetPriority(EXTI0_1_IRQn, 0); /* Setting priority */
    NVIC_EnableIRQ(EXTI0_1_IRQn); /* Enabling the interrupt function */

    while(1) { } /* Endless loop */
    return 0;
}
```

4. Problem 3



In this problem I will connect two external interrupt buttons which are connected to different groups. Each interrupt operation will light their own respectable led for 10 seconds.

C Code starts with setting the led delay value and led delay functions. Chose PA0 for my first interrupt button, and PB0 for the led corresponding to it's led lighting function. Chose PA4 from IRQ 4_15 for the second button input because of pins PA2 and PA3 being absent on the test board and task requirements requesting us of two different interrupt handler groups. Next thing on my C Code is those said interrupt handlers. Override the original interrupt handler functions with my own, which light their own designated external leds, wait 10 seconds and turn the leds off.



```c
/* main.c
    Berat Kizilarmut
    Problem 3, two button interrupt */
#include "stm32g0xx.h"
#define LEDDELAY    16000000 /* 10 Second Delay */

void delay(volatile uint32_t s)
{
    /* Set as volatile to prevent optimization */
    for(; s>0; s--);
}

void EXTI0_1_IRQHandler (void) { /* Overriding default interrupt handler func */
    GPIOB->ODR |= (1U); /* Turning the led on */
    delay(LEDDELAY); /* Delaying for 10 seconds */
    GPIOB->ODR &= ~(1U); /* Turning the led off */
    EXTI->RPR1 |= (1U << 0);
}

void EXTI4_15_IRQHandler (void) {
    GPIOB->ODR |= (2U); /* Turning the led on */
    delay(LEDDELAY); /* Delaying for 10 seconds */
    GPIOB->ODR &= ~(2U); /* Turning the led off */
    EXTI->RPR1 |= (1U << 4);
}

int main(void) { /* Overriding default interrupt handler func */
    /* Setting up the buttons and interruptions */
    /* Enable GPIOA clock */
    RCC->IOPENR |= (1U);
    /* Setup PA0 and PA4 as input */
    /* Used PA4 rather than PA2 or PA3 because pins are absent on the test board
 */
    GPIOA->MODER &= ~(771U);

    /* Setting up the interrupt operation for PA0 */
    EXTI->RTSR1 |= (1U); /* Setting the trigger as rising edge */
    EXTI->EXTICR[0] &= ~(1U << 8*0); /* EXTICR 0 for 0_1 */
    EXTI->IMR1 |= (1U << 0); /* Interrupt mask register */
    EXTI-
>RPR1 |= (1U << 0); /* Rising edge pending register, Clearing pending PA0 */
    NVIC_SetPriority(EXTI0_1_IRQn, 0); /* Setting priority */
    NVIC_EnableIRQ(EXTI0_1_IRQn); /* Enabling the interrupt function */

    /* Setting up the interrupt operation for PA4 */
    EXTI->RTSR1 |= (1U << 4); /* Setting the trigger as rising edge */
    EXTI->EXTICR[1] &= ~(1U << 0); /* EXTICR[1] and offset 0 for PA4 */
    EXTI->IMR1 |= (1U << 4); /* Interrupt mask register */
    EXTI-
>RPR1 |= (1U << 4); /* Rising edge pending register, Clearing pending PA4 */
    NVIC_SetPriority(EXTI4_15_IRQn, 1); /* Setting priority */
    NVIC_EnableIRQ(EXTI4_15_IRQn); /* Enabling the interrupt function */

    /* Setting the output leds */
    /* Enable GPIOB clock */
    RCC->IOPENR |= (2U);
    /* Setup PB0 and PB1 as output */
    GPIOB->MODER &= ~(15U);
    GPIOB->MODER |= (5U);

    while(1) { } /* Endless loop */
    return 0;
}
```

5. References

1. RM0444 STM32G0x1 Reference manual, ST Microelectronics