GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC335

MICROPROCESSORS LAB

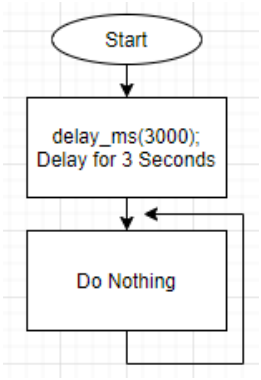
LAB 4 Report

Prepared by
171024086 Berat KIZILARMUT

1. Introduction

In this lab we will be going over timers in microcontrollers. We will be completing five different task each focusing on different important elements we've learned in timers course. First, we will design a delay function by implementing SysTick Exception. Second, we will design a led toggler which changes the toggle speed on button press. Third, we will design a count up timer that counts up to ten thousand. Four, we will experiment with watchdog timers. And lastly we will combine three and four.

2. Problem 1



In this task, a SysTick exception will be created with one millisecond intervals. Using this one millisecond interval a delay_ms function will be created which will delay the operations for a given millisecond amount, in my code 3000 milliseconds.

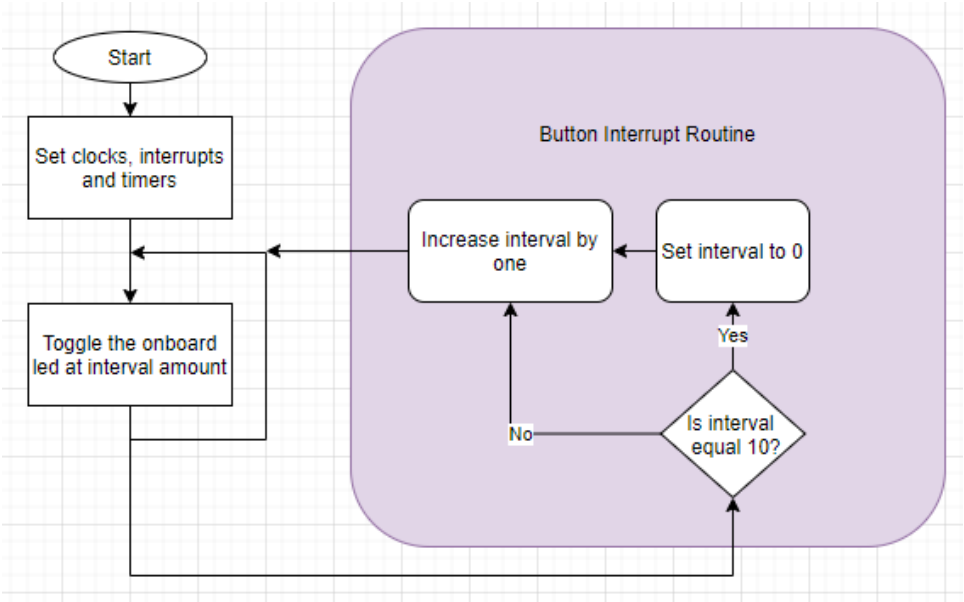
Explaining my code, firstly I've created a delaycount variable to count the systick count. After that I created my delay_ms function as volatile to prevent it being optimized out, set the delaycount as zero there to reset the counter. Continuously checked if the millisecond input is same as the delaycount. After 3 seconds have past, microprocessor returns to doing nothing.

```c
/* main.c Problem 1, millisecond delayer
    Berat Kizilarmut */
#include "stm32g0xx.h"
volatile uint32_t delaycount = 0; /* Delay counter, counts every millisecond */
void delay_ms(volatile uint32_t s) /* Set as volatile to prevent optimization */
{
    delaycount = 0; /* Set counter as zero */
    while(1) /* Continuously checks the counter */
    {
        if(delaycount == s)
        return;
    }
}

void SysTick_Handler(void)
{
    delaycount++; /* SysTick increases the count every millisecond */
}

int main(void) { /* Overriding default interrupt handler func */
    SysTick_Config(16000); /* 16 MHz Clock speed, 16000 ticks mean a millisecond */
    delay_ms(3000); /* Delay 3000 ms, 3 seconds */
    while(1) { } /* Endless loop */
    return 0;
}
```

Software wise accuracy of the delay can be tested with integrated debugger timers, or another delay function like I used prior to this that counts down from a given value in a branch loop and compare the results.

Hardware wise accuracy can be tested with leds lighting up during the countdown and using actual timing equipment like chronometers or video recordings to analyse the accuracy.

3. Problem 2

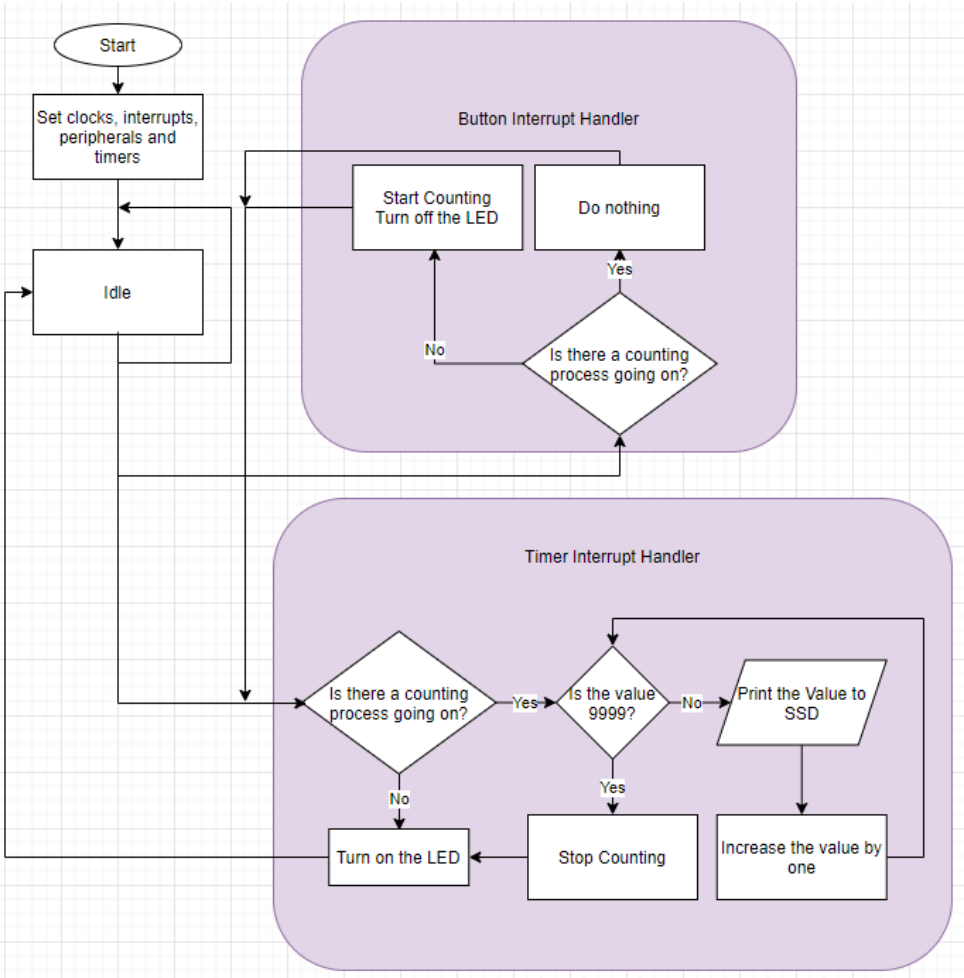


In this task I will be implementing an external interrupt button and a general-purpose timer to flick the onboard led3 on 1 second intervals at first. At each button interrupt flicker time will increase by 1 second up until 10 seconds at which point it will reset back to 1 second intervals.

Explaining my code, beginning from external button interrupt setup; set the GPIOA clock to connect the button on PA0, set the MODER to 00 set up as input. Set the interrupt operation for PA0 via EXTI. Set the trigger as rising edge, control register mux as 0 to use for EXTI0_1. Set the interrupt as the highest priority and enabled the interrupt function. On the EXTI0_1_IRQHandler function itself which I override the default with, I check for the interval amount, if the interval amount is 10 it will reset the interval to 1, if it's not 10 it will increase the interval amount by one. Set the Auto Reload Register of TIM1 to interval * 16000, which is 16k because of 16 MHz, 16k multiplied by the PSC which is 1k equals to 16M. After these operations, clear the pending status to return.

Next up looking at the general-purpose timer, TIM1, I configured a initialization function for it. On this said function I enable the TIM1 clock, cleared the control register, zeroed the internal counter, updated the interrupt enabler, set the PSC to 1k and Auto Reload Register to 16k because of 16 MHz clock speed. On the timer interrupt function TIM1_BRK_UP_TRG_COM_IRQHandler, I toggle the onboard LED3, which is on PC6 and was set up in main, via XOR operation.

```c
/* main.c Problem 2, blink timer
    Berat Kizilarmut */
#include "stm32g0xx.h"
volatile uint32_t interval = 1; /* Setting the second interval counter */

void EXTI0_1_IRQHandler (void) { /* Button interrupt */
    if(interval == 10) /* if the interval is 10, return to 1 second intervals */
    interval = 0; /* Setting it zero here, because line below will increase it once
*/
    interval++; /* Increase the interval time */
    TIM1->ARR = (16000)*(interval); /* Multiply it with 16k to achieve a second */
    EXTI->RPR1 |= (1U << 0); /* Clear pending status */
}

void TIM1_BRK_UP_TRG_COM_IRQHandler (void) /* Configuring TIM1 */
{
    GPIOC->ODR ^= (1U << 6); /* Toggle PC6 */
    TIM1->SR &= ~(1U << 0); /* Clear pending status */
}

void init_timer1(void) /* Setting TIM1 */
{
    RCC->APBENR2 |= (1U << 11); /* Enable TIM1 Clock */
    TIM1->CR1 = 0; /* Clearing the control register */
    TIM1->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM1->CNT = 0; /* Zero the counter */
    TIM1->PSC = 999; /* 1000 since we're starting from 0 */
    TIM1->ARR = 16000; /* For 16 MHz */
    TIM1->DIER |= (1U << 0); /* Updating interrupt enabler */
    TIM1->CR1 |= (1U << 0); /* Enable TIM1 */
    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 3); /* Setting lowest priority */
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn); /* Enabling interrupt */
}

int main(void) {
    /* Setting up the external button and interrupt */
    /* 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 highest priority */
    NVIC_EnableIRQ(EXTI0_1_IRQn); /* Enabling the interrupt function */

    /* Setting the onboard LED3, set default as PC6 */
    /* Enable GPIOC clock */
    RCC->IOPENR |= (1U << 2);
    /* Set PC6 as output */
    GPIOC->MODER &= ~(3U << 12); /* Setting bits 13 and 12 as zero */
    GPIOC->MODER |= (1U << 12); /* Setting bit 12 as one */

    init_timer1(); /* Calling the timer initializer */

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

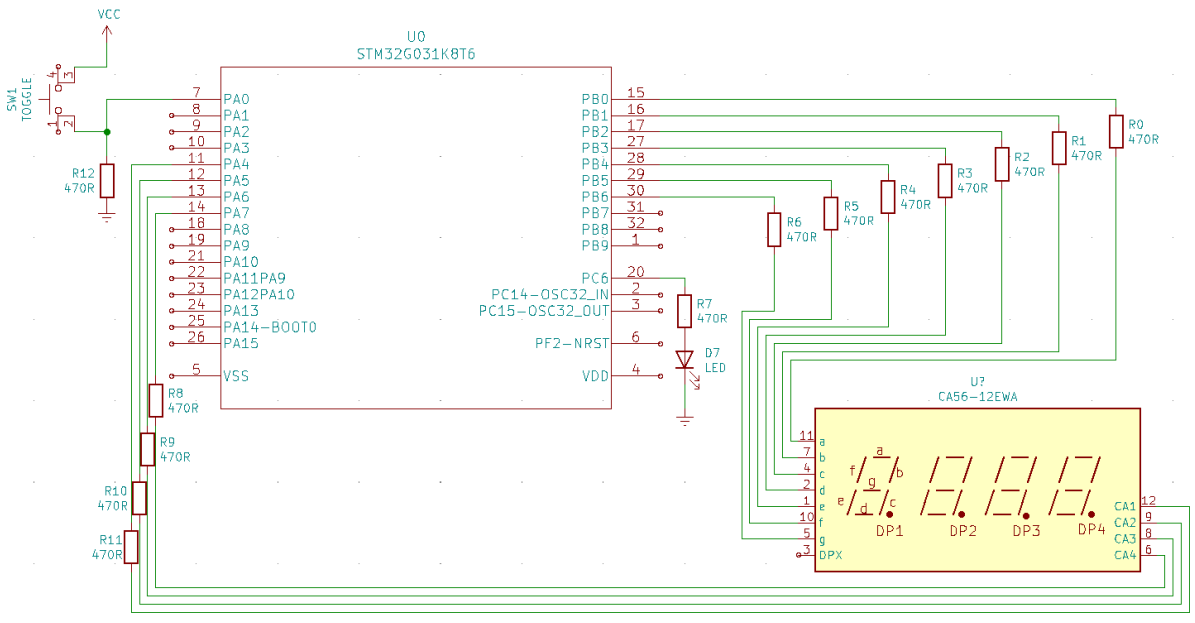


4. Problem 3

In this task I will design a counting up mechanism that prints the counting process to a 4 Digit Seven Segment Display. This design will implement the external interrupt button process and general-purpose timer from Problem 2 and expand it.

Explaining my code, firstly I've set my external button interrupt and general-purpose timer exactly the same way I explained on the previous problem. Only difference is I've changed PSC and ARR value to arrange the timing. Implemented counting state checkpoints to button interrupt handler to check if a counting process is going on when button is pressed. Called the printing functions in the timer interrupt handler, refreshed the printing process with a for loop to increase the brightness of the SSD. Timer interrupt handler changes the state to idle when number 9999 is reached on counting. Implemented the said printing functions which decode the decimal value to its digit points and print the value accordingly.

Connected the Seven Segment Display. Used pins PA4 to PA7 as Digit pins of the SSD, used pins PB0 to PB6 as A B C D E F G pins of the SSD. Digit pins are working active high, A B C D E F G pins are working active low. Utilized the onboard led connected to PC6 as the indicator led which lights up on idle mode, turns off during the counting process.



```c
/* main.c Problem 3, SSD Counter
    Berat Kizilarmut */
#include "stm32g0xx.h"
volatile uint32_t counter = 0; /* Setting the counter */
volatile uint32_t counting_state = 1; /* Setting the state tracker */
/* Zero for idle, one for counting */

void ssd_clear(void) /* Clear the display */
{
    GPIOA->ODR &= ~(15U << 4); /* Clear Digits */
    GPIOB->ODR |= (127U); /* Clear A B C... */
}

void ssd_printdigit(uint32_t s) /* Prints the values depending on the number */
{
    switch(s)
    {
        case 0: /* Printing 0 */
            GPIOB->ODR &= (0xC0U);
            break;

        case 1: /* Printing 1 */
            GPIOB->ODR &= (0xF9U);
            break;

        case 2: /* Printing 2 */
            GPIOB->ODR &= (0xA4U);
            break;

        case 3: /* Printing 3 */
            GPIOB->ODR &= (0xB0U);
            break;

        case 4: /* Printing 4 */
            GPIOB->ODR &= (0x99U);
            break;

        case 5: /* Printing 5 */
            GPIOB->ODR &= (0x92U);
            break;

        case 6: /* Printing 6 */
            GPIOB->ODR &= (0x82U);
            break;

        case 7: /* Printing 7 */
            GPIOB->ODR &= (0xF8U);
            break;

        case 8: /* Printing 8 */
            GPIOB->ODR &= (0x80U);
            break;

        case 9: /* Printing 9 */
            GPIOB->ODR &= (0x90U);
            break;
    }
}

void ssd_print(uint32_t value)
{
    /* Gather the digit by digit values using integer rounding */
    uint32_t d1=0, d2=0, d3=0, d4=0;
    d1=value/1000;
    value=value-(d1*1000);
    d2=value/100;
```

```
    value=value-(d2*100);
    d3=value/10;
    value=value-(d3*10);
    d4=value;
    ssd_clear();
    /* Print Thousands Digit */
    GPIOA->ODR |= (1U << 4); /* Set D1 High */
    ssd_printdigit(d1); /* Print Value to D1 */
    ssd_clear();
    /* Print Hundreds Digit */
    GPIOA->ODR |= (1U << 5); /* Set D2 High */
    ssd_printdigit(d2); /* Print Value to D2 */
    ssd_clear();
    /* Print Tens Digit */
    GPIOA->ODR |= (1U << 6); /* Set D3 High */
    ssd_printdigit(d3); /* Print Value to D3 */
    ssd_clear();
    /* Print Ones Digit */
    GPIOA->ODR |= (1U << 7); /* Set D4 High */
    ssd_printdigit(d4); /* Print Value to D4 */
    ssd_clear();
}

void EXTI0_1_IRQHandler (void) { /* Button interrupt */
    if(counting_state == 1)
    {
        EXTI->RPR1 |= (1U << 0); /* Clear pending status */
    }
    else
    {
        GPIOC->ODR |= (1U << 6); /* Turn on PC6 */
        counter=0;
        counting_state=1;
        EXTI->RPR1 |= (1U << 0); /* Clear pending status */
    }
}

void TIM1_BRK_UP_TRG_COM_IRQHandler (void) /* Configuring TIM1 */
{
    if(counter == 9999)
    {
        GPIOC->ODR |= (1U << 6); /* Turn on PC6 */
        counting_state = 0;
    }
    else
    {
        GPIOC->ODR &= ~(1U << 6); /* Turn off PC6 */
    for(int i=0; i<25; i++) /* Refreshing the display again and again to make it visible
 */
    {
        ssd_print(counter);
    }
    counter++;
    }
    TIM1->SR &= ~(1U << 0); /* Clear pending status */
}

void init_timer1(void) /* Setting TIM1 */
{
    RCC->APBENR2 |= (1U << 11); /* Enable TIM1 Clock */
    TIM1->CR1 = 0; /* Clearing the control register */
    TIM1->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM1->CNT = 0; /* Zero the counter */
    TIM1->PSC = 7999;
    TIM1->ARR = 1;
    TIM1->DIER |= (1U << 0); /* Updating interrupt enabler */
```

```
    TIM1->CR1 |= (1U << 0); /* Enable TIM1 */
    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 3); /* Setting lowest priority */
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn); /* Enabling interrupt */
}

int main(void) {
    /* Setting up the external button and interrupt */
    /* 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 highest priority */
    NVIC_EnableIRQ(EXTI0_1_IRQn); /* Enabling the interrupt function */

    /* Setting the onboard LED3, set default as PC6 */
    /* Enable GPIOC clock */
    RCC->IOPENR |= (1U << 2);
    /* Set PC6 as output */
    GPIOC->MODER &= ~(3U << 12); /* Setting bits 13 and 12 as zero */
    GPIOC->MODER |= (1U << 12); /* Setting bit 12 as one */

    /* Setting up the Seven Segment Display*/
    /* Enable GPIOB clock */
    RCC->IOPENR |= (1U << 1);
    /* Set PB0 to PB6 as output to use as A B C D E F G pins of the SSD */
    GPIOB->MODER &= ~(65535U); /* Setting first 16 bits as zero */
    GPIOB->MODER |= (21845U); /* Setting odd bits as one */
    /* Set PA4 to PA7 as output to use as D1 D2 D3 D4 pins of the SSD */
    GPIOA->MODER &= ~(65280U); /* Setting bits 8th to 16th as zero */
    GPIOA->MODER |= (21760U); /* Setting odd bits as one from 8th to 16th */

    init_timer1(); /* Calling the timer initializer */

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

5. References

1. ARM v6-M Architecture Reference Manual, ARM
2. UM2591 User Manual STM32G0 Nucleo-32 board (MB1455), ST Microelectronics
3. RM0444 Reference Manual, STM32G0x1 advanced Arm®-based 32-bit MCUs, ST Microelectronics