1. INTRODUCTION:

The objective of this lab is to get you communicate with MCU from PC, and utilize bidirectional data transmission and parsing. You will use C language for the problems unless some parts require inline assembly. Use blinky project from stm32go repo as the starting point for your problems.

2. PROBLEMS:

2.1. Problem 1:

In this problem, you will work on implementing a simple tone generator utilizing Timer, PWM and External Interrupt modules and use a keypad, a speaker, and 7SDs.

-Connect a keypad to your microcontroller, pick a block of notes, and assign a tone for each key on the keypad.

-One of the keys should be silence (rest).

-Tones will be played using PWM by changing the period at 50% duty cycle.

-Design an amplifier, and connect a speaker with adjustable gain using a pot.

-When a key is pressed, the relevant tone should play indefinitely.

-Connect your 7SD to display the tone that is being played. You can display the frequency being played. (440, 480,...) OR you can display the tone being played (A4, B4, C4, ...). ,

Theoretical Research:

Pulse-width modulation:

Pulse width modulation (PWM), or pulse-duration modulation (PDM), is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load. Along with maximum power point tracking (MPPT), it is one of the primary methods of reducing the output of solar panels to that which can be utilized by a battery.[1] PWM is particularly suited for running inertial loads such as motors, which are not as easily affected by this discrete switching, because their inertia causes them to react slowly. The PWM switching frequency has to be high enough not to affect the load, which is to say that the resultant waveform perceived by the load must be as smooth as possible.

The rate (or frequency) at which the power supply must switch can vary greatly depending on load and application. For example, switching has to be done several times a minute in an electric stove; 120 Hz in a lamp dimmer; between a few kilohertz (kHz) and tens of kHz for a motor drive; and well into the tens or hundreds of kHz in audio amplifiers and computer power supplies. The main advantage of PWM is that power loss in the switching devices is very low. When a switch is off there is practically no current, and when it is on and power is being transferred to the load, there is almost no voltage drop across the switch. Power loss, being the product of voltage and current, is thus in both cases close to zero. PWM also works well with digital controls, which, because of their on/off nature, can easily set the needed duty cycle. PWM has also been used in certain communication systems where its duty cycle has been used to convey information over a communications channel.

In electronics, many modern microcontrollers (MCUs) integrate PWM controllers exposed to external pins as peripheral devices under firmware control by means of internal programming interfaces. These are commonly used for direct current (DC) motor control in robotics and other applications.

Duty cycle

A **duty cycle** or **power cycle** is the fraction of one period in which a signal or system is active. Duty cycle is commonly expressed as a percentage or a ratio. A period is the time it takes for a signal to complete an on-and-off cycle. As a formula, a duty cycle (%) may be expressed as:

$$D = \frac{PW}{T} \times 100\%$$ [2]

Equally, a duty cycle (ratio) may be expressed as:

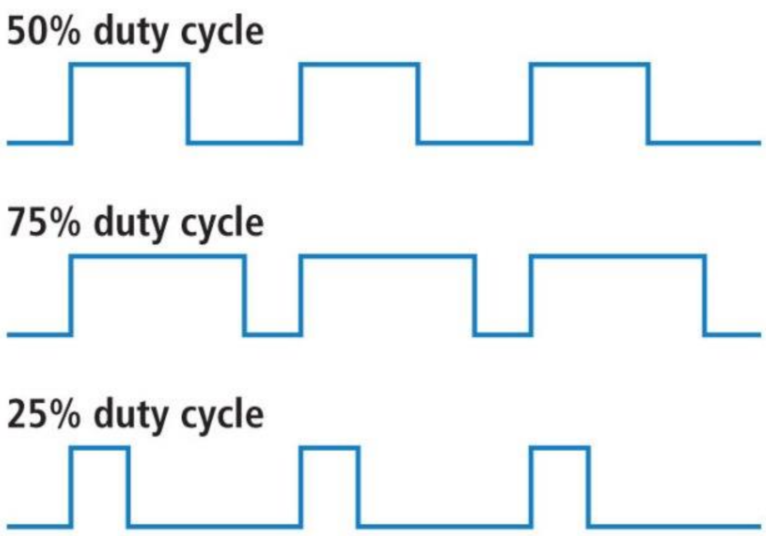
$$D = \frac{PW}{T}$$

where is the duty cycle, is the pulse width (pulse active time), and is the total period of the signal. Thus, a 60% duty cycle means the signal is on 60% of the time but off 40% of the time. The "on time" for a 60% duty cycle could be a fraction of a second, a day, or even a week, depending on the length of the period.

Duty cycles can be used to describe the percent time of an active signal in an electrical device such as the power switch in a switching power supply or the firing of action potentials by a living system such as a neuron.

The **duty factor** for periodic signal expresses the same notion, but is usually scaled to a maximum of one rather than 100%.[

The duty cycle can also be notated as



2.2. Code:

```c
#include "bsp.h"


void BSP_system_init(void){

	__disable_irq();

	init_timer1();

	__enable_irq();


}

void pwm_init(){

	RCC->IOPENR |= (1U << 0);

	GPIOA->MODER &= ~(3U << 2*6);
	GPIOA->MODER |= (2U << 2*6);

	// choose AF2 from mux
	GPIOA->AFR[0] &= ~(0xFU << 2*6);
	GPIOA->AFR[0] |= (2U << 2*6);

	TIM2->CCR3  = 800;

	TIM2->CCMR1 |= (0x6U << 4); // mode 1 enable
	TIM2->CCMR1 |= (1U << 3);
	TIM2->CCER |= (1U << 8);

}



//void init_timer2(){
//
//
//	RCC->IOPENR |= (1U << 0);
//	RCC->APBENR1 |= (1U<< 0);// enable time2 module clock
//
//	//setup PA6 as AF2
//	GPIOA->MODER &= ~(3U << 2*6);
//	GPIOA->MODER |= (2U << 2*6);
//
//	// choose AF2 from mux
//	GPIOA->AFR[0] &= ~(0xFU << 2*6);
//	GPIOA->AFR[0] |= (2U << 2*6);
//
//	TIM2->CR1=0;// zero out the control register just in case
//	TIM2->CR1 |= (1U << 7); // ARPE
//	TIM2->CNT=0;// zero out counter
//
//	/*1 second interrupt	*/
```

```c
//
//  TIM2->PSC = 9;
//  TIM2->ARR = 1600;
//  TIM2->CCR3  = 800;
//
//  TIM2->CCMR1 |= (0x6U << 4); // mode 1 enable
//  TIM2->CCMR1 |= (1U << 3);
//  TIM2->CCER |= (1U << 8);
//
//
//
//  TIM2->DIER |= (1 << 0);// update interrupt enable
//  TIM2->CR1 |= (1 << 0);//    tÄ±m1 enable
//
//  NVIC_SetPriority(TIM2_IRQn,3);
//  NVIC_EnableIRQ(TIM2_IRQn);
//
//
//
//}

void init_timer1(){

    RCC->APBENR1 |= (1U<< 0);// enable time2 module clock

    TIM2->CR1=0;// zero out the control register just in case
    TIM2->CR1 |= (1U << 7); // ARPE
    TIM2->CNT=0;// zero out counter

    /*1 second interrupt     */

    TIM2->PSC = 9;
    TIM2->ARR = 1600;

    TIM2->DIER |= (1 << 0);// update interrupt enable
    TIM2->CR1 |= (1 << 0);//    tÄ±m1 enable

    NVIC_SetPriority(TIM2_IRQn,3);
    NVIC_EnableIRQ(TIM2_IRQn);

}

void TIM2_IRQHandler(void){

    pwm_init();
    TIM2->SR &= ~(1U << 0); //clear update status register.

}
```

```c
#ifndef BSP_H_
#define BSP_H_

#include "stm32g0xx.h"

void delay(volatile unsigned int);

void BSP_system_init();

void pwm_init();
void init_timer1();
//void init_timer2();

#endif
```

```c
/*
 * display.c
 *
 *  Created on: Dec 19, 2020
 *      Author: Mehmet Akif/171024027
 *
 *      description:In this section, necessary pins of
 *      the SSD have been activated in order for the lid ssd.
 *      Next,With the display function, the burning of the digits
 *      at the same time is provided by flashing the digits invisibly.
 *      and then we display the character from printdigit func
 */

#include "display.h"

extern void main();



void init_SSD(){
    GPIOB->MODER &= ~(3U << 2*1);
    GPIOB->MODER |= (1U << 2*1);//PB1 is output

    GPIOB->MODER &= ~(3U << 2*3);
    GPIOB->MODER |= (1U << 2*3);//PB3 is output

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

    GPIOB->MODER &= ~(3U << 2*7);
    GPIOB->MODER |= (1U << 2*7);//PB7 is output

    GPIOA->MODER &= ~(3U << 2*0);
    GPIOA->MODER |= (1U << 2*0);//PA0 is output

    GPIOA->MODER &= ~(3U << 2*1);
    GPIOA->MODER |= (1U << 2*1);//PA1 is output

    GPIOA->MODER &= ~(3U << 2*4);
    GPIOA->MODER |= (1U << 2*4);//PA4 is output

    GPIOA->MODER &= ~(3U << 2*5);
    GPIOA->MODER |= (1U << 2*5);//PA5 is output

    GPIOA->MODER &= ~(3U << 2*6);
    GPIOA->MODER |= (1U << 2*6);//PA6 is output

    GPIOA->MODER &= ~(3U << 2*7);
    GPIOA->MODER |= (1U << 2*7);//PA7 is output

    GPIOA->MODER &= ~(3U << 2*11);
    GPIOA->MODER |= (1U << 2*11);//PA11 is output

    GPIOA->MODER &= ~(3U << 2*12);
    GPIOA->MODER |= (1U << 2*12);//PA12 is output


}

void display_SSD(){

    static int i = 0;
```

```
        if(Display.Inv == 1){
            invalid_SSD();

        }
        else if(Display.Oflw == 1){
            overflow_SSD();

        }

        if(i == 1){
            GPIOA->ODR |= (1U << 7); //PA7
            GPIOB->ODR &= ~(1U << 6); //PB6
            GPIOB->ODR &= ~(1U << 7); //PB7
            GPIOB->ODR &= ~(1U << 1); //PB1
            printDigit_SSD(Display.Digits[0]);
            GPIOA->ODR |= ( 1U << 6); // PA6
            if(Display.dot == 0) GPIOA->ODR &= ~( 1U << 6);
        }
        else if(i == 10){
            GPIOA->ODR &= ~(1U << 7); //PA7
            GPIOB->ODR |= (1U << 6); //PB6
            GPIOB->ODR &= ~(1U << 7); //PB7
            GPIOB->ODR &= ~(1U << 1); //PB1
            printDigit_SSD(Display.Digits[1]);
            GPIOA->ODR |= ( 1U << 6); // PA6
            if(Display.dot == 1) GPIOA->ODR &= ~( 1U << 6);
        }

        else if(i == 20){
            GPIOA->ODR &= ~(1U << 7); //PA7
            GPIOB->ODR &= ~(1U << 6); //PB6
            GPIOB->ODR |= (1U << 7); //PB7
            GPIOB->ODR &= ~(1U << 1); //PB1
            printDigit_SSD(Display.Digits[2]);
            GPIOA->ODR |= ( 1U << 6); // PA6
            if(Display.dot == 2) GPIOA->ODR &= ~( 1U << 6);
        }
        else if(i == 30){
            GPIOA->ODR &= ~(1U << 7); //PA7
            GPIOB->ODR &= ~(1U << 6); //PB6
            GPIOB->ODR &= ~(1U << 7); //PB7
            GPIOB->ODR |= (1U << 1); //PB1
            printDigit_SSD(Display.Digits[3]);
            GPIOA->ODR |= ( 1U << 6); // PA6
            if(Display.dot == 3) GPIOA->ODR &= ~( 1U << 6);
        }
        else if(i == 40) i = 0;

        i++;




}

void printDigit_SSD(uint8_t x){

        switch(x){
        case 0:     //0

            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
```

```c
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR |=  ( 1U << 11); // PA11

            break;

        case 1:     //1
            GPIOB->ODR |= ( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR |= ( 1U << 4); // PA4
            GPIOA->ODR |= ( 1U << 5); // PA5
            GPIOA->ODR |= ( 1U << 12); // PA12
            GPIOA->ODR |= ( 1U << 11); // PA11

            break;

        case 2:     //2
            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR |=  ( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR |=  ( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11

            break;

        case 3:     //3

            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR |=  ( 1U << 5); // PA5
            GPIOA->ODR |=  ( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11

            break;

        case 4:     //4
            GPIOB->ODR |=  ( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR |=  ( 1U << 4); // PA4
            GPIOA->ODR |=  ( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11

            break;

        case 5:     //5

            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR |=  ( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR |=  ( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11
```

```c

            break;

        case 6:     //6
            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR |=  ( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11

            break;

        case 7:     //7

            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR |=  ( 1U << 4); // PA4
            GPIOA->ODR |=  ( 1U << 5); // PA5
            GPIOA->ODR |=  ( 1U << 12); // PA12
            GPIOA->ODR |=  ( 1U << 11); // PA11

            break;

        case 8:     //8

            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11

            break;

        case 9:     //9
            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR |=  ( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11

            break;

        case 10://A

            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR |=  ( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11



            break;
```

```c

        case 11://B

            GPIOB->ODR |=  ( 1U << 3); // PB3
            GPIOA->ODR |=  ( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11


            break;

        case 12://C
            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR |=  ( 1U << 0); // PA0
            GPIOA->ODR |=  ( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR |=  ( 1U << 11); // PA11

            break;

        case 13://D
            GPIOB->ODR |=  ( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR |=  ( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11

            break;

        case 14://E
            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR |=  ( 1U << 0); // PA0
            GPIOA->ODR |=  ( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11


            break;

        case 15: //F
            GPIOB->ODR &= ~( 1U << 3); // PB3
            GPIOA->ODR |=  ( 1U << 0); // PA0
            GPIOA->ODR |=  ( 1U << 1); // PA1
            GPIOA->ODR |=  ( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11
            break;


        case 30: //u
            GPIOB->ODR |=  ( 1U << 3); // PB3
            GPIOA->ODR |=  ( 1U << 0); // PA0
```

```
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR |=  ( 1U << 12); // PA12
            GPIOA->ODR |=  ( 1U << 11); // PA11
            break;

        case 31: //L
            GPIOB->ODR |=  ( 1U << 3); // PB3
            GPIOA->ODR |=  ( 1U << 0); // PA0
            GPIOA->ODR |=  ( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR &= ~( 1U << 12); // PA12
            GPIOA->ODR |=  ( 1U << 11); // PA11
            break;

        case 32: //n
            GPIOB->ODR |=  ( 1U << 3); // PB3
            GPIOA->ODR |=  ( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR |=  ( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR |=  ( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11
            break;

        case 33: //D
            GPIOB->ODR |=  ( 1U << 3); // PB3
            GPIOA->ODR &= ~( 1U << 0); // PA0
            GPIOA->ODR &= ~( 1U << 1); // PA1
            GPIOA->ODR &= ~( 1U << 4); // PA4
            GPIOA->ODR &= ~( 1U << 5); // PA5
            GPIOA->ODR |=  ( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11
            break;

        case 34: // negative sign
            GPIOB->ODR |= ( 1U << 3); // PB3
            GPIOA->ODR |= ( 1U << 0); // PA0
            GPIOA->ODR |= ( 1U << 1); // PA1
            GPIOA->ODR |= ( 1U << 4); // PA4
            GPIOA->ODR |= ( 1U << 5); // PA5
            GPIOA->ODR |= ( 1U << 12); // PA12
            GPIOA->ODR &= ~( 1U << 11); // PA11
            break;

        case 35: // space
            GPIOB->ODR |= ( 1U << 3); // PB3
            GPIOA->ODR |= ( 1U << 0); // PA0
            GPIOA->ODR |= ( 1U << 1); // PA1
            GPIOA->ODR |= ( 1U << 4); // PA4
            GPIOA->ODR |= ( 1U << 5); // PA5
            GPIOA->ODR |= ( 1U << 12); // PA12
            GPIOA->ODR |= ( 1U << 11); // PA11
            break;
        }
}


void displaychar_SSD(uint8_t x){
    Display.dot = 5;
```

```
switch(x){
case 0: //-
    Display.Digits[0] = 34;
    Display.Digits[1] = 35;
    Display.Digits[2] = 35;
    Display.Digits[3] = 35;
    break;
case 1: //'A'
    Display.Digits[3] = 10;
    Display.Digits[0] = 35;
    Display.Digits[2] = 35;
    Display.Digits[1] = 35;
    break;
case 2: //'B'
    Display.Digits[3] = 11;
    Display.Digits[1] = 35;
    Display.Digits[2] = 35;
    Display.Digits[0] = 35;
    break;
case 3:  //'C'
    Display.Digits[3] = 12;
    Display.Digits[1] = 35;
    Display.Digits[2] = 35;
    Display.Digits[0] = 35;
    break;
case 4: //'D'
    Display.Digits[3] = 13;
    Display.Digits[1] = 35;
    Display.Digits[2] = 35;
    Display.Digits[0] = 35;
    break;
case 5:  //'E'
    Display.Digits[3] = 14;
    Display.Digits[2] = 35;
    Display.Digits[1] = 35;
    Display.Digits[0] = 35;
    break;
case 6: //'EA'
    Display.Digits[3] = 14;
    Display.Digits[2] = 10;
    Display.Digits[1] = 35;
    Display.Digits[0] = 35;
    break;
case 7: //'EB'
    Display.Digits[3] = 14;
    Display.Digits[2] = 11;
    Display.Digits[1] = 35;
    Display.Digits[0] = 35;
    break;
case 8: //'EC'
    Display.Digits[3] = 14;
    Display.Digits[2] = 12;
    Display.Digits[1] = 35;
    Display.Digits[0] = 35;
    break;
case 9: //'ED'
    Display.Digits[3] = 14;
    Display.Digits[2] = 13;
    Display.Digits[1] = 35;
    Display.Digits[0] = 35;
    break;
case 10:    //'EE'
```

```
            Display.Digits[3] = 14;
            Display.Digits[2] = 14;
            Display.Digits[1] = 35;
            Display.Digits[0] = 35;
            break;
        case 11:    //'EEA'
            Display.Digits[3] = 14;
            Display.Digits[2] = 14;
            Display.Digits[1] = 10;
            Display.Digits[0] = 35;
            break;
        case 12:    //'EEB'
            Display.Digits[3] = 14;
            Display.Digits[2] = 14;
            Display.Digits[1] = 11;
            Display.Digits[0] = 35;
            break;
        case 13:    //'EEC'
            Display.Digits[3] = 14;
            Display.Digits[2] = 14;
            Display.Digits[1] = 12;
            Display.Digits[0] = 35;
            break;
        case 14:    //'EED'
            Display.Digits[3] = 14;
            Display.Digits[2] = 14;
            Display.Digits[1] = 13;
            Display.Digits[0] = 35;
            break;
        case 20: // 0uFL
            Display.Digits[0] = 31;
            Display.Digits[1] = 15;
            Display.Digits[2] = 30;
            Display.Digits[3] = 0;
            break;
        case 21: // InuD
            Display.Digits[0] = 33;
            Display.Digits[1] = 30;
            Display.Digits[2] = 32;
            Display.Digits[3] = 1;

            break;
    }
}

void displayID_SSD(){
    Display.Digits[0]= 7;
    Display.Digits[1]= 2;
    Display.Digits[2]= 7;
    Display.Digits[3]= 1;


}

void overflow_SSD(){

    displaychar_SSD(20);

    Display.Oflw = 0;
    calculation.current_process=0;
    calculation.x=0;
    calculation.y=0;
```

```
    calculation.result=0;

}

void invalid_SSD(void){

    displaychar_SSD(21);

    Display.Inv = 0;
    calculation.current_process=0;
    calculation.x=0;
    calculation.y=0;
    calculation.result=0;

}

void utility_SSD(float var){

    int number = (int)var;

    float i = 0.0;

    if((number < 0) & (number >= -999)){
        Display.sign = 1;
        i = -1.0;
        Display.dot = 0;
        if(number >= -99){
            i = -10.0;
            Display.dot = 1;
            if(number >= -9){
                i = -100.0;
                Display.dot = 2;
            }
        }
    }
    else if((number >= 0) & (number <= 9999)){
        Display.sign = 0;
        i = 1.0;
        Display.dot = 0;
        if(number <= 999){
            i = 10.0;
            Display.dot = 1;
            if(number <= 99){
                i = 100.0;
                Display.dot = 2;
                if(number <= 9){
                    i = 1000.0;
                    Display.dot = 3;
                }
            }
        }
    }


    number = (int)(var * i);

    int temp = number / 10;
    Display.Digits[0] = (uint8_t)(number - (temp*10));

    temp = number / 100;
    Display.Digits[1] = (uint8_t)((number - (temp * 100)) / 10);

```

```
    temp = number / 1000;
    Display.Digits[2] = (uint8_t)((number - (temp * 1000)) / 100);

    temp = number / 10000;
    Display.Digits[3] = (uint8_t)((number - (temp * 10000)) / 1000);

    // negative sign
    if (Display.sign) Display.Digits[3] = 34;

}
```

```c
/*
 * display.h
 *
 *  Created on: Dec 19, 2020
 *      Author: Mehmet Akif/171024027
 */

#ifndef DISPLAY_H_
#define DISPLAY_H_

#include "main.h"
#include "bsp.h"

typedef struct{
    uint8_t Digits[4];
    uint8_t Oflw:1;
    uint8_t sign:1;
    uint8_t dot:3;
    uint8_t Inv:1;
}SSD;

/*
 * Display struct keep the digits and
 * overflow, sign, dot, invalid bits
 */
SSD Display;

/*
 * initiation for keypad pins
 */
void init_SSD();

/*
 * This function ensures that the digits on the display
 * are lit by quickly flashing the digits.
 */
void display_SSD();

/*
 * the cases which are inside of this func show that
 * how to display the character
 */
void printDigit_SSD(uint8_t);

void displayID_SSD();

/*
 * when the result bigger than 9999 or less than -999
 * display shows that OuFL
 */
void overflow_SSD();

/*
 * when the operation is invalid i.e. 3/0 or sqrt(-2))
 * display shows that Invd
 */
void invalid_SSD();

/*
 * separates the incoming result into digit
 * and we can see that if number is negative or
 * not through sign bit
```

```
 */
void utility_SSD(float var);

/*
 * It determines which character should be lit on which
 * digit by assigning case values to digit.
 */
void displaychar_SSD(uint8_t x);

#endif /* DISPLAY_H_ */

```

```c
/*
 * keypad.c
 *
 *  Created on: Dec 19, 2020
 *      Author: Mehmet Akif/171024027
 *      description:In this section, necessary pins of
 *      the keypad have been activated in order for the keypad
 *      buttons to receive data. Next, the interrupt was created
 *      for the buttons. Thanks to this interrupt, when the button
 *      is pressed, it is processed according to priority.
 *      After determining which character the received data is from,
 *      it was sent to the display function for printing. It was sent to
 *      the calculation function for the necessary operations.
 */
#include "keypad.h"

/*to the reach delay func*/
extern void delay_ms(volatile unsigned int);

void keypad_init(void){

       /* Enable GPIOB and GPIOA clock */
        RCC->IOPENR |= (1U << 1);
        RCC->IOPENR |= (1U << 0);


        /* Setup PA8,PB9,PB5 and PB4 as output (rows)*/
        GPIOA->MODER &= ~(3U << 2*8);
        GPIOA->MODER |= (1U << 2*8);//PA8 is output

        GPIOB->MODER &= ~(3U << 2*9);
        GPIOB->MODER |= (1U << 2*9);//PB9 is output

        GPIOB->MODER &= ~(3U << 2*5);
        GPIOB->MODER |= (1U << 2*5);//PB5 is output

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



        /* Setup PA9,PB0,PB2 and PB8  as input(colums) */
        GPIOA->MODER &= ~(3U << 2*9);// PA9 is input
        GPIOA->PUPDR |= (2U << 2*9); // Pull-down mode

        GPIOB->MODER &= ~(3U << 2*0);//PB0 is input
        GPIOB->PUPDR |= (2U << 2*0); // Pull-down mode

        GPIOB->MODER &= ~(3U << 2*2);//PB2 is input
        GPIOB->PUPDR |= (2U << 2*2); // Pull-down mode

        GPIOB->MODER &= ~(3U << 2*8);//PB8 is input
        GPIOB->PUPDR |= (2U << 2*8); // Pull-down mode


        /*setup interrupts for inputs*/
        EXTI->EXTICR[2] |=(0U << 8*1);//PA9
        EXTI->EXTICR[0] |=(1U << 0);//PB0
        EXTI->EXTICR[0] |=(1U << 2*8);//PB2
        EXTI->EXTICR[2] |=(1U << 0);//PB8


```

```c
        /*rising edge*/
        EXTI->RTSR1 |= (1U << 9);// 9th pin
        EXTI->RTSR1 |= (1U << 0);// 0th pin
        EXTI->RTSR1 |= (1U << 2);// 2th pin
        EXTI->RTSR1 |= (1U << 8);// 8th pin


        /* MASK*/
        EXTI->IMR1 |= (1U << 9);// 9th pin
        EXTI->IMR1 |= (1U << 0);// 0th pin
        EXTI->IMR1 |= (1U << 2);// 2th pin
        EXTI->IMR1 |= (1U << 8);// 8th pin


        /*NVIC*/
        NVIC_SetPriority(EXTI0_1_IRQn,0);
        NVIC_EnableIRQ(EXTI0_1_IRQn);

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

        NVIC_SetPriority(EXTI4_15_IRQn,0);
        NVIC_EnableIRQ(EXTI4_15_IRQn);

}
/* interrut for PB0*/
void EXTI0_1_IRQHandler(void){
    if (EXTI->RPR1 & (1U << 0)){// check if pending register equal 1

        clear_rows_keypad();
        /* make PA8 enable*/
        GPIOA->ODR ^=( 1U << 8);
        if ((GPIOB->IDR >> 0) & 1){//check if PB0 equal 1
            /* #=(F) character*/
            Keypad_data(15);

        }
        /*make PA8 disable*/
        GPIOA->ODR ^=( 1U << 8); // PA8

        /* make PB9 enable*/
        GPIOB->ODR ^=( 1U << 9); // PB9
        if ((GPIOB->IDR >> 0) & 1){
            /* 9 character*/
            Keypad_data(9);

        }
        /* make PB9 disable*/
        GPIOB->ODR ^=( 1U << 9); // PB9

        /* make PB5 enable*/
        GPIOB->ODR ^=( 1U << 5); // PB5
        if ((GPIOB->IDR >> 0) & 1){
            /* 6 character*/
            Keypad_data(6);

        }
        /* make PB5 disable*/
        GPIOB->ODR ^=( 1U << 5); // PB5

        /* make PB4 enable*/
        GPIOB->ODR ^=( 1U << 4); // PB4
```

```
        if ((GPIOB->IDR >> 0) & 1){
            /* 3 character*/
            Keypad_data(3);

        }
        /* make PB4 disable*/
        GPIOB->ODR ^=( 1U << 4); // PB4


        set_rows_keypad();
        /*clear interrupt for clear pending register */
        EXTI->RPR1 |= (1U << 0);
    }
}

/* interrut for PB2*/
void EXTI2_3_IRQHandler(void){

    if (EXTI->RPR1 & (1U << 2)){// check if pending register equal 1

        clear_rows_keypad();
        /*make PA8 enable*/
        GPIOA->ODR ^=( 1U << 8); // PA8
        if ((GPIOB ->IDR >> 2) & 1){//check if PB2 equal 1
            /* D character*/
            Keypad_data(13);

        }
        /*make PA8 disable*/
        GPIOA->ODR ^=( 1U << 8); // PA8

        /* make PB9 enable*/
        GPIOB->ODR ^=( 1U << 9); // PB9
        if ((GPIOB ->IDR >> 2) & 1){
            /* C character*/
            Keypad_data(12);

        }
        /* make PB9 disable*/
        GPIOB->ODR ^=( 1U << 9); // PB9

        /* make PB5 enable*/
        GPIOB->ODR ^=( 1U << 5); // PB5
        if ((GPIOB ->IDR >> 2) & 1){
            /* B character*/
            Keypad_data(11);

        }
        /* make PB5 disable*/
        GPIOB->ODR ^=( 1U << 5); // PB5

        /* make PB4 enable*/
        GPIOB->ODR ^=( 1U << 4); // PB4
        if ((GPIOB ->IDR >> 2) & 1){
            /* A character*/
            Keypad_data(10);

        }
        /* make PB4 disable*/
        GPIOB->ODR ^=( 1U << 4); // PB4


```

```
        set_rows_keypad();
        /*clear interrupt for clear pending register */
        EXTI->RPR1 |= (1U << 2);
    }
}

/* interrut for PB8 and PA9*/
void EXTI4_15_IRQHandler(void){

            /*interrut for PB8*/
        if (EXTI->RPR1 & (1U << 8)){// check if pending register equal 1
            clear_rows_keypad();
            /*make PA8 enable*/
            GPIOA->ODR ^=( 1U << 8); // PA8
            if ((GPIOB ->IDR >> 8) & 1){//check if PB8 equal 1
                /* *(E) character*/
                Keypad_data(14);

            }
            /*make PA8 disable*/
            GPIOA->ODR ^=( 1U << 8); // PA8

            /* make PB9 enable*/
            GPIOB->ODR ^=( 1U << 9); // PB9
            if ((GPIOB ->IDR >> 8) & 1){
                /* 7 character*/
                Keypad_data(7);

            }
            /* make PB9 disable*/
            GPIOB->ODR ^=( 1U << 9); // PB9

            /* make PB5 enable*/
            GPIOB->ODR ^=( 1U << 5); // PB5
            if ((GPIOB ->IDR >> 8) & 1){
                /* 4 character*/
                Keypad_data(4);

            }
            /* make PB5 disable*/
            GPIOB->ODR ^=( 1U << 5); // PB5

            /* make PB4 enable*/
            GPIOB->ODR ^=( 1U << 4); // PB4
            if ((GPIOB ->IDR >> 8) & 1){
                /* 1 character*/
                Keypad_data(1);

            }
            /* make PB4 disable*/
            GPIOB->ODR ^=( 1U << 4); // PB4


            set_rows_keypad();
            /*clear interrupt for clear pending register */
            EXTI->RPR1 |= (1U << 8);
        }

        /*interrut for PA9*/
        if (EXTI->RPR1 & (1U << 9)){// check if pending register equal 1
            clear_rows_keypad();
            /*make PA8 enable*/
```

```c
            GPIOA->ODR ^=( 1U << 8); //check if PA8 equal 1
            if ((GPIOA ->IDR >> 9) & 1){
                /* 0 character*/
                Keypad_data(0);

            }
            /*make PA8 disable*/
            GPIOA->ODR ^=( 1U << 8); // PA8

            /* make PB9 enable*/
            GPIOB->ODR ^=( 1U << 9); // PB9
            if ((GPIOA ->IDR >> 9) & 1){
                /* 8 character*/
                Keypad_data(8);

            }
            /* make PB9 disable*/
            GPIOB->ODR ^=( 1U << 9); // PB9

            /* make PB5 enable*/
            GPIOB->ODR ^=( 1U << 5); // PB5
            if ((GPIOA ->IDR >> 9) & 1){
                /* 5 character*/
                Keypad_data(5);

            }
            /* make PB5 disable*/
            GPIOB->ODR ^=( 1U << 5); // PB5

            /* make PB4 enable*/
            GPIOB->ODR ^=( 1U << 4); // PB4
            if ((GPIOA ->IDR >> 9) & 1){
                /* 2 character*/
                Keypad_data(2);

            }
            /* make PB4 disable*/
            GPIOB->ODR ^=( 1U << 4); // PB4


            set_rows_keypad();

            /*clear interrupt for clear pending register */
            EXTI->RPR1 |= (1U << 9);
        }

}


void clear_rows_keypad(void){
        /*clearing the rows here*/
        GPIOA->ODR &= ~(1U << 8);//PA8
        GPIOB->ODR &= ~(1U << 9);//PB9
        GPIOB->ODR &= ~(1U << 5);//PB5
        GPIOB->ODR &= ~(1U << 4);//PB4
}

void set_rows_keypad(void){
        /*seting the rows here*/
        GPIOA->ODR |= (1U << 8);//PA8
        GPIOB->ODR |= (1U << 9);//PB9
        GPIOB->ODR |= (1U << 5);//PB5
```

```c
        GPIOB->ODR |= (1U << 4);//PB4

}

void Keypad_data(uint8_t a){

    static int i = 0;

    /*
    *if the digits are already full,
    *new number key presses are ignored
    *by counter int i.
    **/
    if ((a<10) & (i<4)){
        calculation.x = (calculation.x * 10) + (float)a;
        utility_SSD(calculation.x);
        i++;
    }
    /*
    *if ABCDE pressed,
    *calculation mode will be selected
    **/
    else if ((a>9) & (i<=4)){
        i = 4;

        if(a == 10){
            calculation.current_process = Addition;
            displaychar_SSD(Addition);
            i = 7;
        }
        else if(a == 11){
            calculation.current_process = Substraction;
            displaychar_SSD(Substraction);
            i = 7;
        }
        else if(a == 12){
            calculation.current_process = Multiplacation;
            displaychar_SSD(Multiplacation);
            i = 7;
        }
        else if(a == 13){
            calculation.current_process = Division;
            displaychar_SSD(Division);
            i = 7;
        }
        else if(a == 14){
            calculation.current_process = E;
            displaychar_SSD(E);
            i = 5;
        }
    }
/*
 * if E Key is pressed, scientific mode on
 * and expect another keypress
 */
    else if (i == 5){

        if(a == 10){
            calculation.current_process = Log;
            displaychar_SSD(Log);
            i = 11;
        }
```

```c
        else if(a == 11){
            calculation.current_process = Ln;
            displaychar_SSD(Ln);
            i = 11;
        }
        else if(a == 12){
            calculation.current_process = Sqrt;
            displaychar_SSD(Sqrt);
            i = 11;
        }
        else if(a == 13){
            calculation.current_process = Pow2;
            displaychar_SSD(Pow2);
            i = 11;
        }
        else if(a == 14){
            calculation.current_process = EE;
            displaychar_SSD(EE);
            i = 6;
        }
    }
    /*
     * if EE Key is pressed, trigonometric mode on
     * and expect another keypress
     */
    else if (i == 6){

        if(a == 10){
            calculation.current_process = Sin;
            displaychar_SSD(Sin);
            i = 11;
        }
        else if(a == 11){
            calculation.current_process = Cos;
            displaychar_SSD(Cos);
            i = 11;
        }
        else if(a == 12){
            calculation.current_process = Tan;
            displaychar_SSD(Tan);
            i = 11;
        }
        else if(a == 13){
            calculation.current_process = Cot;
            displaychar_SSD(Cot);
            i = 11;
        }
        else if(a == 14){
            calculation.x = 3.141;
            utility_SSD(calculation.x);
            i = 4;
        }
    }
    /*
     * if ABCD pressed, requires another number
     * for calculation
     */
    else if((a<10) & (i >= 7) & (i < 11)){
        calculation.y = (calculation.y * 10) + (float)a;
        utility_SSD(calculation.y);
        i++;
    }
```

```
    /*
     * F key is for equal
     */
    else if ((i == 11) | (a == 15)){

        calculate();
        i = 0;
        calculation.x = calculation.result;
        calculation.y = 0;
        calculation.current_process = 0;
        utility_SSD(calculation.x);
    }
}
```

```c
/*
 * keypad.h
 *
 *  Created on: Dec 19, 2020
 *      Author: Mehmet Akif/171024027
 */

#ifndef KEYPAD_H_
#define KEYPAD_H_

#include "calc.h"
#include"main.h"

/*Keypad related function*/
void keypad_init();     //initiation for keypad pins
void clear_rows_keypad(); // set 0  keypad rows
void set_rows_keypad(); // set 1 keypad rows


/* taken data from button which is pressed
 and figure out which button is this*/
void Keypad_data(uint8_t a);


#endif /* KEYPAD_H_ */
```

Problem 2:

In this problem, you will be working with reading and logging MPU6050 IMU sensor data utilizing Timer, I2C, and UART modules and use MPU6050, and 24LC512 EEPROM.

- Write your I2C routines to read / write multiple data. You should have four functions: single read, single write, multi read, multi write. Multi read and write deal with multiple bytes.
- To ensure the data is correct, send your results over UART to PC with You should read all sensor data and send them all as the example below:
  1 AX: 0.12, AY: 0.53, .., GX: 1.32, ...
- Sample the sensors every 10 ms, and write the data values to EEPROM. You should first start with writing and reading single bytes. Once the operation is completed successfully, work on your way to write and read multiple bytes.
- EEPROM and MPU6050 should be sharing the same I2C bus. Check the IMU board for pull-up resisitors. If it includes pull-up resistors, you should not need to add another set of pull-up resistors.
- Once you press an external button, data collection should start, and once it collects 10 seconds of data, it should stop, and an LED should light up to display data is ready on EEPROM.
- When the LED is on (meaning there is data on the EEPROM) pressing the button will transmit all the data over UART to your PC.
- You can optionally save this to a file and/or plot your results using Python or Matlab.

Theoretical Research:

I2C tutorial:

This I2C tutorial shows you how the I2C protocol or more correctly written I^2C (sometimes written as IIC) stands for Inter IC Communication and is intended for very short distance communication between ICs on a single PCB. It gives you a fully defined protocol for data transfer between multiple devices over two wires.

In this **I2C tutorial** you will learn all about the 2 wire I2C serial protocol; How easy it is to use, how it works and when to use it.

The I2C protocol is used in a huge range of chips - just a few examples from this site include the DS1307 (RTC), SSD1306 (OLED Display), MCP23017 (Serial expander). The protocol allows you to connect many devices to a single set of two wires, and then communicate individually with each device.

This I2C tutorial shows you how the I2C protocol works at the physical bit level discussing single master mode (a single controlling device) which is the most common use for I2C in a small system.

512Kb I2C compatible 2-wire Serial EEPROM:

The Microchip Technology Inc. 24LC512 is a 512Kb (64K x 8) Serial Electrically Erasable PROM (EEPROM), capable of operation across a broad voltage range (2.5V to 5.5V). It has been developed for advanced, low-power applications such as personal communications and data acquisition. This device also has a page write capability of up to 128 bytes of data. This device is capable of both random and sequential reads up to the 512K boundary. Functional address lines allow up to eight devices on the same bus, for up to 4 Mbit address space. This device is available in the standard 8-pin plastic DIP, SOIJ and DFN packages.

Additional Features

- **Reliable EEPROM Memory**
  - 64K x 8 (512 Kbit)
  - 128-Byte Page Write Buffer
  - Page Write Time 5 ms Max.
  - Hardware Write-Protect Pin
  - Factory Programming Available
- **Low Power**
  - Operating voltage 1.7V to 5.5V
  - Read current 400 uA, max.
  - Standby current 1 uA, max.
- **2-Wire Serial Interface, I^2C™ Compatible**
  - Cascadable up to Eight Devices
  - 100 kHz and 400 kHz Clock Compatible
- Pb-Free and RoHS Compliant

1.1. Code:

```c
/*
 * bsp.c
 *
 *  Created on: 22 Ara 2020
 *      Author: Mehmet Akif/171024027
 */
#include "bsp.h"

void BSP_init(){
    __disable_irq();
    init_I2C();
    //SystemCoreClockUpdate();  //contains the system frequency
    init_timer1();

    __enable_irq();
}

void init_timer1(){

    RCC->APBENR2 |= (1U<< 11);// enable time1 module clock

    TIM1->CR1=0;// zero out the control register just in case
    TIM1->CR1 |= (1<<7);    // ARPE
    TIM1->CNT=0;// zero out counter

    /*0.1 ms interrupt   */

    TIM1->PSC=99;
    TIM1->ARR=16;

    TIM1->DIER |= (1 << 0);// update interrupt enable
    TIM1->CR1 |= (1 << 0);//    tım1 enable

    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn,3);
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);

}

void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
{
    TIM1->SR &= ~(1U<<0); //clear update status register

}

void delay_ms(volatile unsigned int s){

    for(int i=s; i>0; i--){
     SysTick_Config(SystemCoreClock / 1000); // 16 MHz / 1000 ile 1 ms elde edildi.
     }
}


```

```c
/*
 * bsp.h
 *
 *  Created on: 22 Ara 2020
 *      Author: Mehmet Akif/171024027
 */

#ifndef BSP_H_
#define BSP_H_

#include "main.h"

void BSP_init();
void init_timer1();

void delay_ms(volatile unsigned int);

#endif /* BSP_H_ */

```

```c
/*
 * eeprom.c
 *
 *  Created on: 29 Ara 2020
 *      Author: mehme
 */

#include "eeprom.h"

void read_eeprom(uint16_t regAddr, uint8_t *data, uint32_t num){
    uint8_t devAddr = 0x00;
    uint8_t arr[2];
    arr[0] = 0;
    arr[1] = 0;
    write_i2c(devAddr, arr, 1);
    read_i2c(devAddr, data, num);
}

void write_eeprom(uint16_t regAddr, uint8_t *data, uint32_t num){
    uint8_t devAddr = 0x00;
    uint8_t arr[2];
    arr[0] = 0;
    arr[1] = 0;
    write_i2c(devAddr, arr, 1);
    write_i2c(devAddr, data, num);
}
```

```c
/*
 * eeprom.h
 *
 *  Created on: 29 Ara 2020
 *      Author: mehme
 */

#ifndef EEPROM_H_
#define EEPROM_H_

#include "main.h"

void read_eeprom(uint16_t regAddr, uint8_t *data, uint32_t num);
void write_eeprom(uint16_t regAddr, uint8_t *data, uint32_t num);

#endif /* EEPROM_H_ */

```

```c
/*
 * mpu6050.c
 *
 *  Created on: 29 Ara 2020
 *      Author: mehme
 */

#include "mpu6050.h"

void read_mpu(uint8_t regAddr, uint8_t *data, uint32_t num){
    uint8_t devAddr = 0x00;
    write_i2c(devAddr, &regAddr, 1);
    read_i2c(devAddr, data, num);

}
void write_mpu(uint8_t regAddr, uint8_t *data, uint32_t num){
    uint8_t devAddr = 0x00;
    write_i2c(devAddr, &regAddr, 1);
    write_i2c(devAddr, data, num);
}

```

```c
/*
 * mpu6050.h
 *
 *  Created on: 29 Ara 2020
 *      Author: mehme
 */

#ifndef MPU6050_H_
#define MPU6050_H_

#include "main.h"

void read_mpu(uint8_t regAddr, uint8_t *data, uint32_t num);
void write_mpu(uint8_t regAddr, uint8_t *data, uint32_t num);


#endif /* MPU6050_H_ */



```

```c
/*
 * I2C.c
 *
 *  Created on: 27 Ara 2020
 *      Author: mehme
 */

#include "I2C.h"


void I2C1_IRQHandler(void){

}


void init_I2C(void){

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

    //setup PB8 as AF6
    GPIOB->MODER &= ~(3U << 2*8);
    GPIOB->MODER |= (2 << 2*8);
    GPIOB->OTYPER |= (1U << 8);

    //choose AF from mux
    GPIOB->AFR[1] &=  ~(0xFU << 4*0);
    GPIOB->AFR[1] |=  (6 << 4*0);

    //setup PB9 as AF6
    GPIOB->MODER &= ~(3U << 2*9);
    GPIOB->MODER |= (2 << 2*9);
    GPIOB->OTYPER |= (1U << 9);

    //choose AF from mux
    GPIOB->AFR[1] &=  ~(0xFU << 4*1);
    GPIOB->AFR[1] |=  (6 << 4*1);

    //enable I2C1
    RCC->APBENR1 |= (1U << 21);

    I2C1->TIMINGR |= (3 << 28); //PRESC
    I2C1->TIMINGR |= (0x13 << 0); //SCLL
    I2C1->TIMINGR |= (0xF << 8); //SCLH
    I2C1->TIMINGR |= (0x2 << 16); //SDADEL
    I2C1->TIMINGR |= (0x4 << 20); //SCLDEL

    I2C1->CR1 = (1U << 0); //PE

    NVIC_SetPriority(I2C1_IRQn, 1);
    NVIC_EnableIRQ(I2C1_IRQn);
}




void write_i2c(uint8_t devAddr, uint8_t *data, uint32_t num){

    //Write operation (Send address and register to read)
    I2C1->CR2 = 0;
    I2C1->CR2 |= ((uint32_t)devAddr << 1); // slave address
    I2C1->CR2 |= (num << 16); // Number of byte
```

```c
    I2C1->CR2 |= (1U << 25); // AUTOEND
    I2C1->CR2 |= (1U << 13); // Generate Start

    for (size_t i=0; i<num; ++i){
        while(!(I2C1->ISR & (1 << 1))); //TXIS
        I2C1->TXDR = data[i];
    }
}


void read_i2c(uint8_t devAddr, uint8_t *data, uint32_t num){

    I2C1->CR2 = 0;
    I2C1->CR2 |= ((uint32_t)devAddr << 1);
    I2C1->CR2 |= (1U << 10); //READ mode
    I2C1->CR2 |= (num << 16); //Number of bytes
    I2C1->CR2 |= (1U << 15); //NACK
    I2C1->CR2 |= (1U << 25); //AUTOEND
    I2C1->CR2 |= (1U << 13); //Generate Start

    for(size_t i=0; i<num; i++){
        while(!(I2C1->ISR & (1 << 2))); // wait until RXNE =1
        data[i] = (uint8_t)I2C1->RXDR;
        }
}


```

```c
/*
 * I2C.h
 *
 *  Created on: 27 Ara 2020
 *      Author: mehme
 */

#ifndef I2C_H_
#define I2C_H_

#include "main.h"


#define MPU6050_ADDRESS         0x68

#define MPU6050_WHO_AM_I        0x75
#define MPU6050_PWR_MGMT_1      0x6B

#define MPU6050_ACCEL_XOUT_H    0x3B
#define MPU6050_ACCEL_XOUT_L    0x3C
#define MPU6050_ACCEL_YOUT_H    0x3D
#define MPU6050_ACCEL_YOUT_L    0x3E

#define MPU6050_GYRO_XOUT_H     0x43
#define MPU6050_GYRO_XOUT_L     0x44

void init_I2C();
void read_i2c(uint8_t devAddr, uint8_t *data, uint32_t num);
void write_i2c(uint8_t devAddr, uint8_t *data, uint32_t num);


typedef struct {
    uint8_t ax;
    uint8_t ay;
    uint8_t az;
    uint8_t gx;
    uint8_t gy;
    uint8_t gz;
}values;


#endif /* I2C_H_ */

```

```c
/*
 * main.c
 *

 */

#include "main.h"






void delay(volatile uint32_t);

int main(void) {

    BSP_init();

    return 0;
}

void delay(volatile uint32_t s) {
    for(; s>0; s--);
}

```

```c
/*
 * main.h
 *
 *  Created on: 27 Ara 2020
 *      Author: mehme
 */

#ifndef MAIN_H_
#define MAIN_H_

#include "stm32g0xx.h"
#include "stdlib.h"
#include "I2C.h"


#endif /* MAIN_H_ */

```

CONCLUSION

The objective of this lab is to get you communicate with MCU from PC, and utilize bidirectional data transmission and parsing. All functions required in Problem 1 are prepared etc. 7SDs, keypad and timer except Pwm module. While preparing the Pwm module, the required registers were assigned and the required pins were activated. But pwm was not observed to work. Where the error happened has not been found. So although everything was ready in problem 1, it could not be completely completed. In Problem 2, briefly requested from us; Reading the data from MPU6050 IMU sensor with I2C and writing to the memor with 24LC512 EEPROM I2C. Then transfer the values in EEPROM to PC via UART. We are expected to transfer data for 10 seconds at 10ms intervals. This corresponds to 1000 pieces of data in 10 seconds. Most of I2C has been completed, but multiple bit reading and writing could not be completed. Necessary research has been done. A struct was created to hold data in EEPROM. However, the systems created could not be successfully combined. Many problems were successfully concluded, but other problems could not be completed because there was not enough time and the self-teach technique was used in this lesson.

References

- https://learn.sparkfun.com/tutorials/pulse-width-modulation/duty-cycle
- https://en.wikipedia.org/wiki/Pulse-width_modulation
- https://www.robot-electronics.co.uk/i2c-tutorial#:~:text=I2C%20Device%20Addressing,be%20from%200%20to%20127.
- https://elinux.org/Interfacing_with_I2C_Devices
- https://www.best-microcontroller-projects.com/i2c-tutorial.html
- https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
- https://www.microchip.com/wwwproducts/en/24LC512