ELECTRONICS ENGINEERING

ELEC335 - MICROPROCESSORS LABORATORY

LAB #6

Muhammet Cemal Eryiğit	1801022024
Şahabettin Alpcan Soydaş	1801022014
Mert Tuncay Firil	1901022285

PROBLEM 1:

In this problem, you will be working on implementing a signal follower.
Attach a signal to one of the pins, capture its value and replay it back using PWM. You should see the original signal back on the oscilloscope

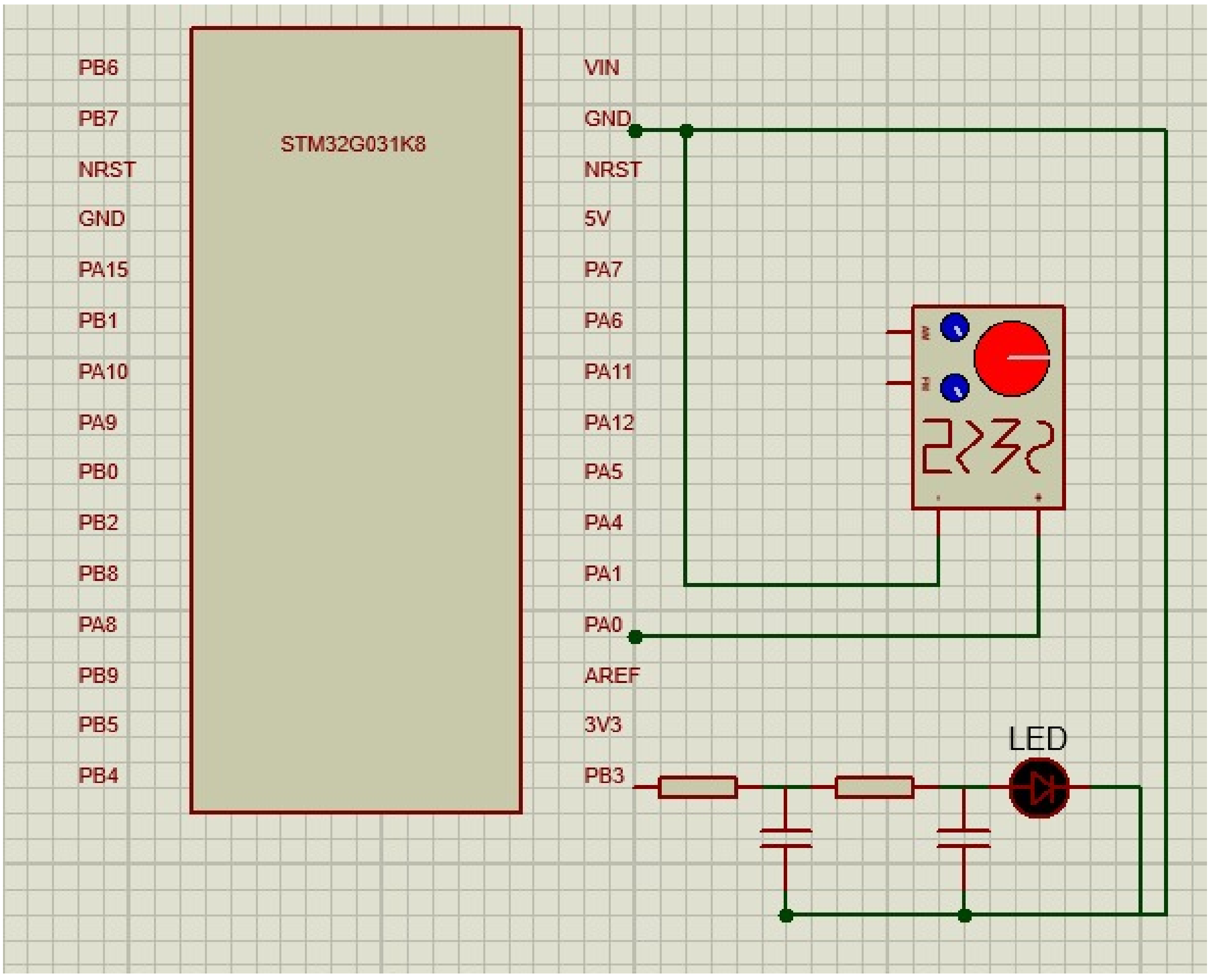

Figure 1.1 Circuit Diagram

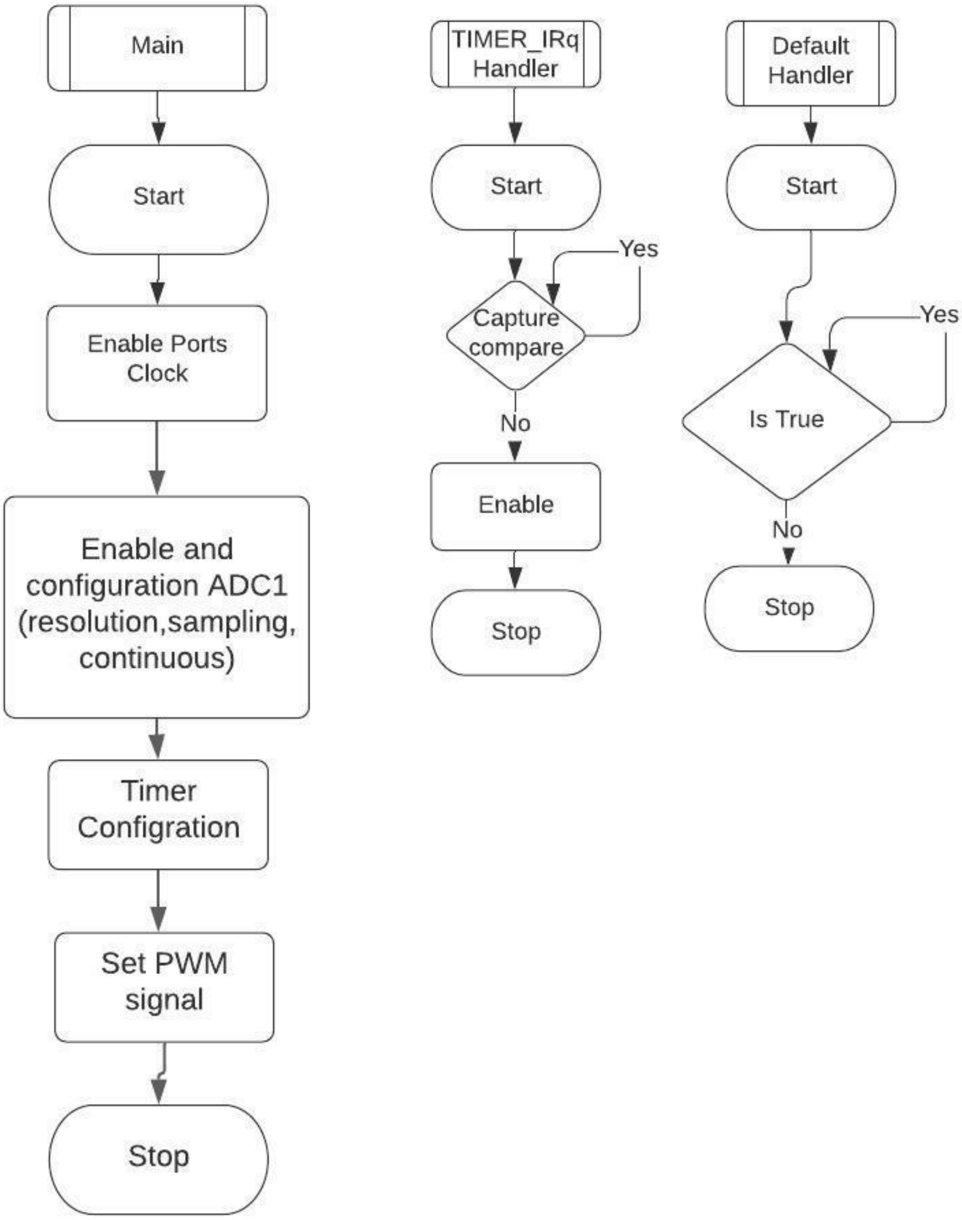

Figure 1.2 Flowchart

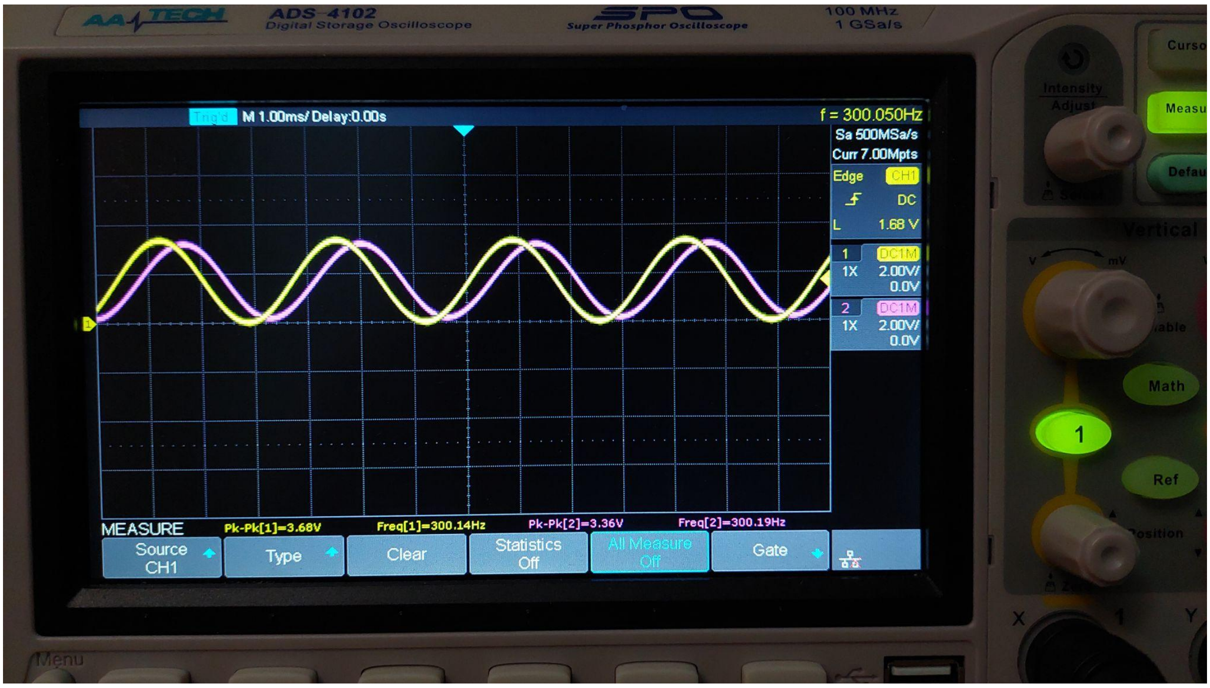

Figure 1.3 300 Hz sine signal input and filtered output (Green: Input from signal generator, Purple: Filtered signal of the signal output from the microprocessor)

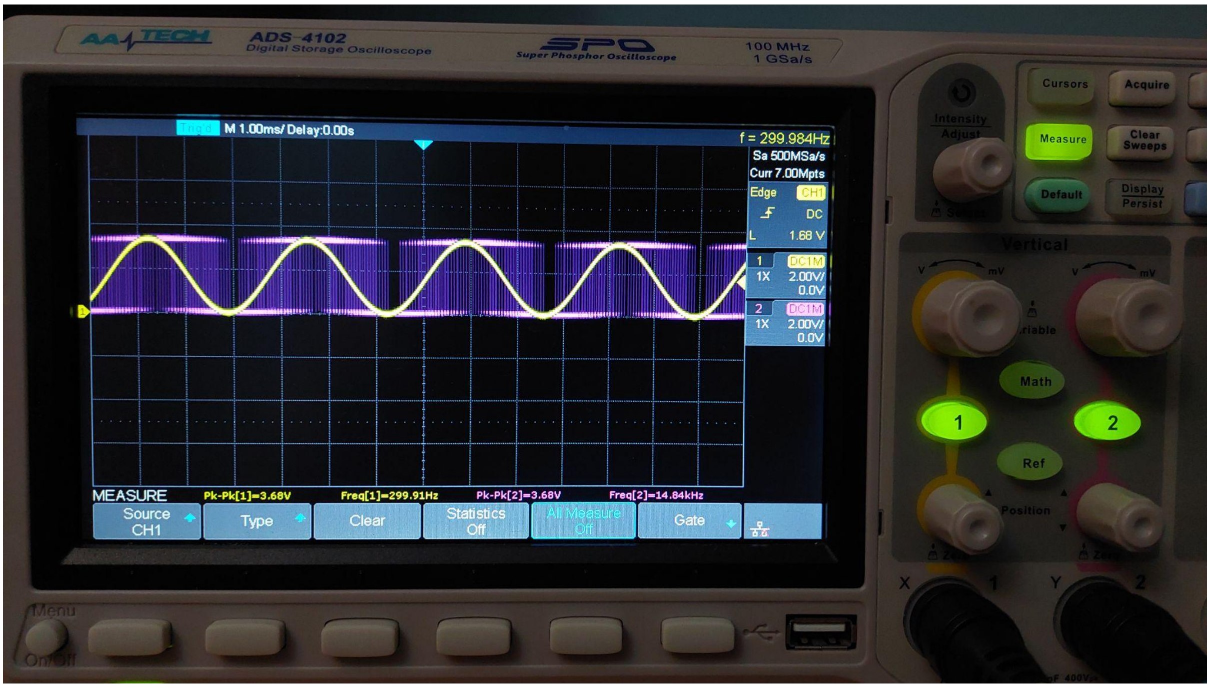

Figure 1.4 300 Hz sine signal input and unfiltered output (Green: Input from signal generator, Purple: Unfiltered signal of the signal output from the microprocessor)

Figure 1.5 300 Hz sine signal input and pwm frequency measurement of the output signal (Green: Input, Purple: Output)

PROBLEM 1 CONCLUSION:

In this problem, the signal from the signal generator is applied as an input to the microprocessor. This signal is produced as a PWM signal by using ADC in a microprocessor. The frequency of this PWM signal was measured as 15.72 kHz and is shown in Figure 1.5.

CODE:

```
#include "stm32g0xx.h"
#include "bsp.h"
#include "system_stm32g0xx.h"



/*PWM Timer */
void TIM2_IRQHandler(void) {
   // update duty (CCR1)
   TIM2->CCR2 = ADC1->DR;

   // Clear update status register
   TIM2->SR &= ~(1U << 0);
}



/* Function to initialize a pwm on PB3 (D13 pin)*/
void init_pwm(){
            // Setup GPIO
              //
```

```
		// Enable GPIOB clock
		RCC->IOPENR |= (1U << 1);
		// Enable TIM2 clock
		RCC->APBENR1 |= RCC_APBENR1_TIM2EN;

		// Set alternate function to 2
		// 3 comes from PB3
		GPIOB->AFR[0] |= (2U << 4*3);
		// Select AF from Moder
		GPIOB->MODER &= ~(3U << 2*3);
		GPIOB->MODER |= (2U << 2*3);

		// zero out the control register just in case
		TIM2->CR1 = 0;

		// Select PWM Mode 1
		TIM2->CCMR1 |= (6U << 12);
		// Preload Enable
		TIM2->CCMR1 |= TIM_CCMR1_OC2PE;

		// Capture compare ch2 enable
		TIM2->CCER |= TIM_CCER_CC2E;

		// zero out counter
		TIM2->CNT = 0;
		TIM2->PSC = 0;
		TIM2->ARR = 1023;

		// zero out duty
		TIM2->CCR2 = 0;

		// Update interrupt enable
		TIM2->DIER |= (1 << 0);

		// TIM1 Enable
		TIM2->CR1 |= TIM_CR1_CEN;

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



/*ADC on PA1 (A1)*/
void init_ADC(){
	//set A1 as analog mode
	setMode('A',1,'A');

	//open ADC clock
	RCC->APBENR2 |= (1U << 20);

	//ADC struct is defined for common mode, we have to use ADC1 struct to change ADC
```

```
register


//disable the ADC as initial just in case
 // this fields are probably zero as initial, it is not mandatory to assign them zero

	ADC1->CR &= ~(1U << 0); // disable the ADC enable bit ADEN
	ADC1->CR &= ~(1U << 1); // disable ADDIS
	ADC1->CR &= ~(1U << 2); // disable ADSTART
	ADC1->CR &= ~(1U << 4); // disable ADSTP
	ADC1->CR &= ~(1U << 31); // disable ADCAL

	//open ADC voltage regulator enable bit
	ADC1->CR |= (1U << 28);
	delay_ms(1); //1ms delay for wait to regulator to regulate the voltage (20 us should be fine,
no need to 1ms)

	//do the ADC calibration ADC calibration
	ADC1->CR |= (1U << 31);
	while(ADC1->CR == (9U << 28));//wait until calibration is done
	ADC1->IER |= (1U << 11); //enable end of the calibration interrupt register (EOCALIE)



	//set the resolution
	ADC1->CFGR1 |= (1U << 3); //set the ADC resolution as 10 bits (max 1023)(CFGR->RES
register (10))

	//configure to continuous mode
	ADC1->CFGR1 &= ~(1U << 16);//disable discontinuous mode if its opened before just in
case
	ADC1->CFGR1 |= (1U << 13);

	//configure sampling rates
	ADC1->SMPR |= (5U << 0);// set the sampling rate 110 mode (sample per 79.5 clock
cycles)
	ADC1->SMPR &= ~(1U << 8);//set smpsel1 register which we're just configured

	//Enable the channels
	ADC1->CHSELR |= (1U << 1);//select channel A1 to read

	ADC1->CR |= (1U << 0); // enable ADC
	while(((ADC1->CR)>>0)  == (1U));
	ADC1 -> CR |= (1U << 2);//ADSTAR1 for ADC

}

int main(void) {

	openClock('A');
	openClock('B');
	init_systick(SystemCoreClock/1000);
	//initilize a pwm to give input signal back
	init_pwm();
```

```
        //initialize the ADC
        init_ADC();


        while(1){


        }

        return 0;
}
```

PROBLEM 2:

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

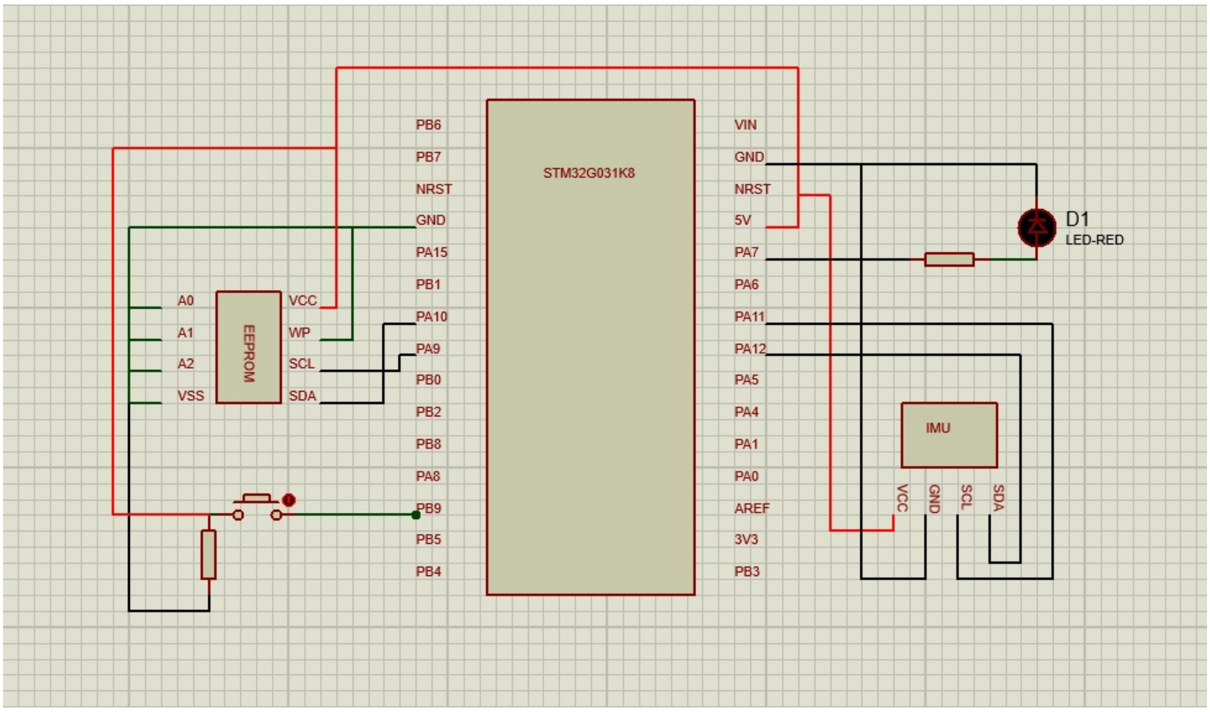

Figure 2.1 Circuit Diagram

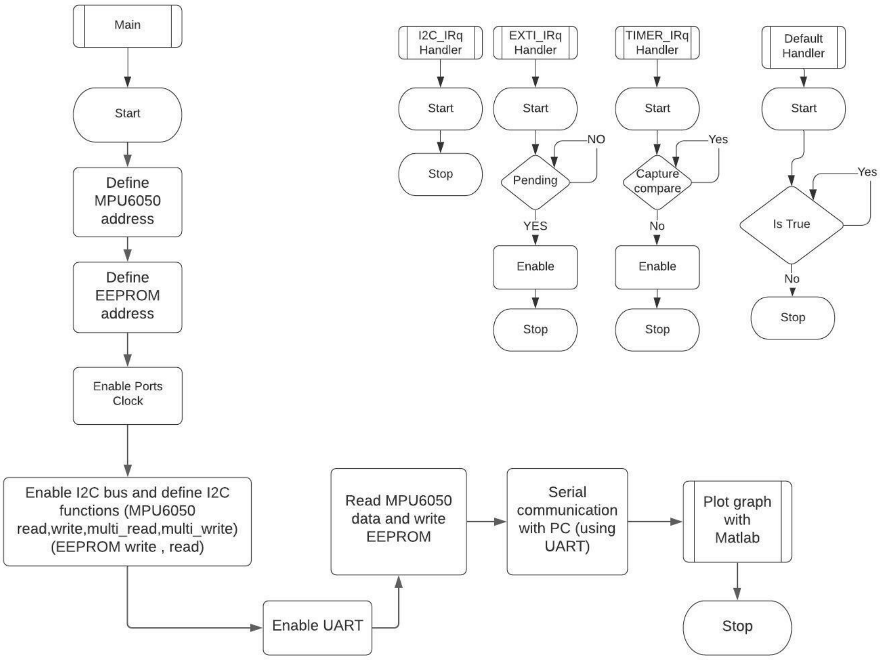

Figure 2.2 Flowchart

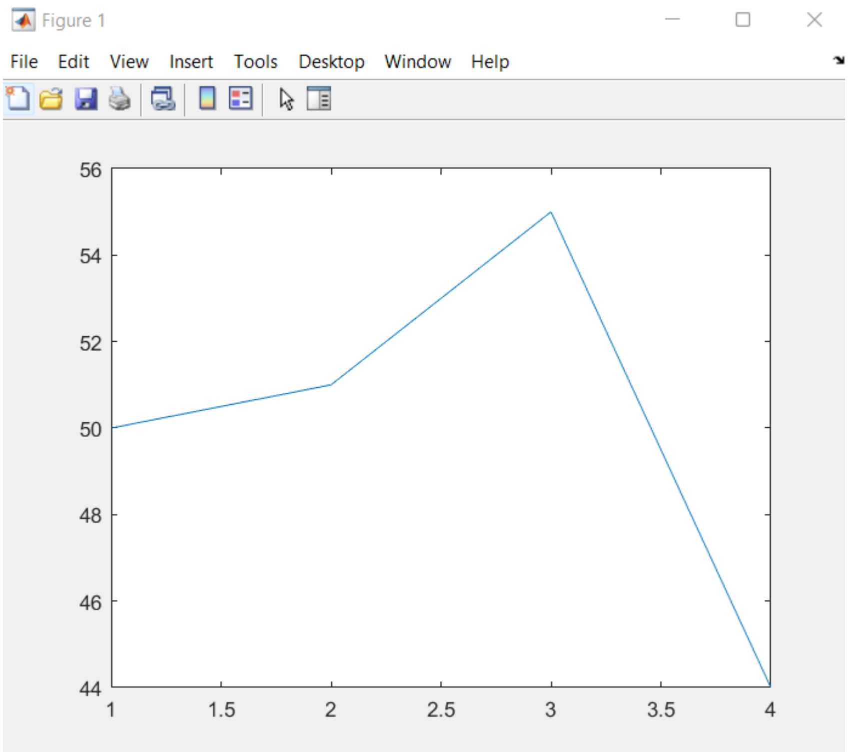

Figure 2.3 First position MPU6050 data

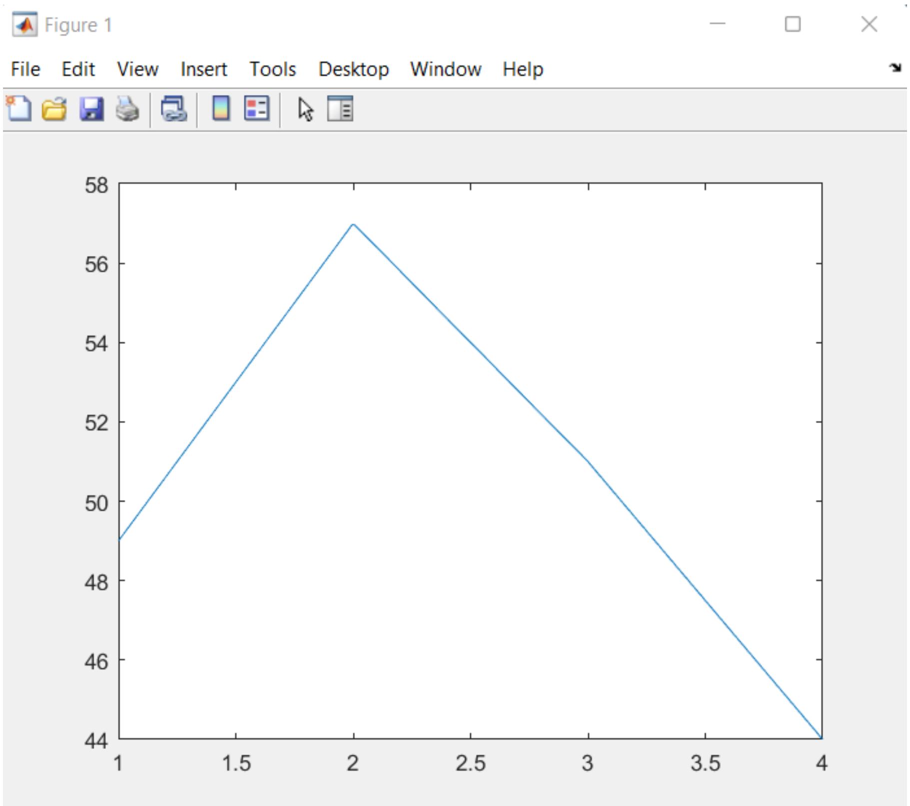

Figure 2.4 Second position MPU6050 data plot

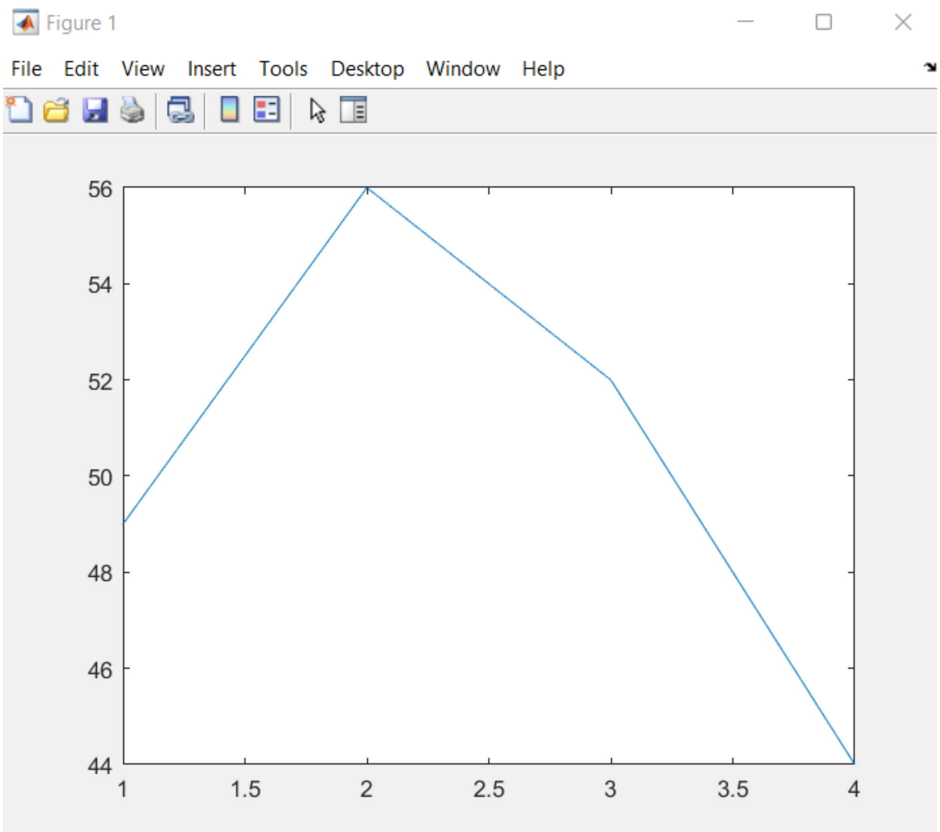

Figure 2.5 Third position MPU6050 data plot

PROBLEM 2 CONCLUSION:

In this problem MPU6050 logging is implemented. I2C read and write functions are defined for MPU6050 and then writing and read functions are defined for EEPROM. Then using UART, logged gyroscope_x , gyroscope_y , accelerometer_x and accelerometer_y data are read from 24LC512 by PC. MPU6050 raw datas(not meaningful data) are plotted using Matlab because floating processes are limited in the Matlab and datas are unsigned 8 bit integer type. When the uint8 type data is divided by 1634.0 there is no meaningful data. Because of that we have to plot only raw sensor data.

MATLAB CODE:

```
clc;
clear all;
close all;
device = serialport("COM6",9600)
data = read(device , 4 , "uint8" );
t = 1:1:4;
plot(t,data);
```

STM32 CODE:

```
#include "stm32g0xx.h"
#include <stdio.h>
// I2C busses == PB8-PB9
#define MPU6050_ADDRESS       0x68
#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
#define MPU6050_GYRO_YOUT_H   0x45
#define MPU6050_GYRO_YOUT_L   0x46
#define BUTTON_DELAY 100000


void GPIO_Config(void);
void print(char *buf);
void printChar(uint8_t c);
int _write(int fd, char *ptr, int len);
void USART_Config(uint16_t baud);
void delay(uint32_t s);
void _read_I2C(uint8_t devAddr,uint16_t memAddr, uint8_t *data, int
size);
void _write_I2C(uint8_t devAddr,uint16_t memAddr, uint8_t *data, int
size);
```

```
uint8_t read_I2C(uint8_t devAddr, uint8_t regAddr);
void write_I2C(uint8_t devAddr, uint8_t regAddr, uint8_t data);
void init_I2C(void);
void multi_Read_I2C(uint8_t devAddr, uint8_t regAddr, uint8_t *data,
uint32_t num);

struct MPU6050_DATA{
      float accel_x;
      float accel_y;
      float gyro_x;
      float gyro_y;
}MPU6050;
uint16_t data;
uint16_t EEPROM_MEMORY = 0x00;
uint8_t EEPROM_ADDRESS = 0x50;

uint16_t gyro_x;
uint16_t accel_x;
uint16_t gyro_y;
uint16_t accel_y;
uint8_t MPU6050_data[4];
uint8_t data_is_back[4];
uint32_t counter = 0;
void button_config(){
      RCC->IOPENR |= (1U << 1);   //Enable GPIOB
      GPIOB->MODER &= ~(3U << 2*0);
      GPIOB->MODER |= (0U << 2*0);  //Enable pb0 as input
      GPIOB->PUPDR |= (2U << 2*0);  //Enable pulldown resistor
      EXTI->EXTICR[0] |= (1U << 8*0);
      EXTI->RTSR1 |= (1U << 0);
      EXTI->IMR1 |= (1U << 0);
      NVIC_EnableIRQ(EXTI0_1_IRQn);

}
void led_config(){
      GPIOB->MODER &= ~(3U << 2*1); //Enable pb1 as output
      GPIOB->MODER |= (1U << 2*1);
}
void EXTI0_1_IRQHandler(){
      if((counter >= BUTTON_DELAY) && (EXTI->RPR1 & (1U << 0) == (1U <<
0))){
            counter = 0;
            GPIOB->ODR |= (1U << 1);
            data = read_I2C(MPU6050_ADDRESS, MPU6050_GYRO_XOUT_L);
            data = data | (read_I2C(MPU6050_ADDRESS,
MPU6050_GYRO_XOUT_H) << 8);
            MPU6050_data[0] = data; //gyro_x
            MPU6050.gyro_x = (float)(data) / (131.0);

            data = read_I2C(MPU6050_ADDRESS, MPU6050_GYRO_YOUT_L);
            data = data | (read_I2C(MPU6050_ADDRESS,
MPU6050_GYRO_YOUT_H) << 8);
            MPU6050_data[1] = data; //gyro_y
```

```
            MPU6050.gyro_y = (float)(data) / (131.0);

            data = read_I2C(MPU6050_ADDRESS, MPU6050_ACCEL_XOUT_L);
            data = data | (read_I2C(MPU6050_ADDRESS,
MPU6050_ACCEL_XOUT_H) << 8);
            MPU6050_data[2] = data; //accel_x
            MPU6050.accel_x = (float)(data) / (16384.0);

            data = read_I2C(MPU6050_ADDRESS, MPU6050_ACCEL_YOUT_L);
            data = data | (read_I2C(MPU6050_ADDRESS,
MPU6050_ACCEL_YOUT_H) << 8);
            MPU6050_data[3] = data;
            MPU6050.accel_y = (float)(data) / (16384.0);

            /*printf("MPU6050 GYRO_X = %f\r\n",MPU6050.gyro_x);
            delay(10000);
            printf("MPU6050 GYRO_Y = %f\r\n",MPU6050.gyro_y);
            delay(10000);
            printf("MPU6050 ACCEL_X = %f\r\n",MPU6050.accel_x);
            delay(10000);
            printf("MPU6050 ACCEL_Y = %f\r\n",MPU6050.accel_y);
            delay(10000);
            */

EEPROM_write_I2C(EEPROM_ADDRESS,EEPROM_MEMORY,&MPU6050_data,4);
            delay(100);
            //printf("EEPROM_WRITTEN_DATA: %d, %d , %d
,%d\r\n",MPU6050_data[0],MPU6050_data[1],MPU6050_data[2],MPU6050_data[3]
);

EEPROM_read_I2C(EEPROM_ADDRESS,EEPROM_MEMORY,&data_is_back,4);
            printf("%d, %d , %d
,%d\r\n",data_is_back[0],data_is_back[1],data_is_back[2],data_is_back[3]
);
            EEPROM_MEMORY += 4;
            delay(1000000);
      }
      EXTI->RPR1 |= (1U << 0);
}


void I2C1_IRQHandler(void){

      //only enters when error
}

int main(void) {

      init_I2C();
      GPIO_Config();
      USART_Config(9600);
      button_config();
      led_config();
```

```
	write_I2C(MPU6050_ADDRESS,  MPU6050_PWR_MGMT_1, 0x00); //disable
sleep mode for MPU6050
	while(1) {
		if(counter <= BUTTON_DELAY){
			counter++;
		}
		GPIOB->ODR &= ~(1U << 1);
    }
    return 0;
}

void init_I2C(void){
	RCC->IOPENR |= (1U << 1);   //Enable GPIOB
	//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); //High register
	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 AF6 from mux
	GPIOB->AFR[1] &= ~(0xFU << 4*1);
	GPIOB->AFR[1] |= (6 << 4*1);

	RCC->APBENR1 |= (1U << 21); //Enable I2C1
	I2C1->CR1 = 0; //RESET CR1
	I2C1->CR1 |= (1U << 7); //ERR1
	I2C1->TIMINGR |= (3U << 28);   //PRESC
	I2C1->TIMINGR |= (0x13U << 0); //SCLL
	I2C1->TIMINGR |= (0xFU << 8);  //SCLH
	I2C1->TIMINGR |= (0x2U << 16); //SDADEL
	I2C1->TIMINGR |= (0x4U << 20); //SCLDEL
	I2C1-> CR1 |= (1U << 0); //PE
	NVIC_SetPriority(I2C1_IRQn, 1);
	NVIC_EnableIRQ(I2C1_IRQn);
}

uint8_t read_I2C(uint8_t devAddr, uint8_t regAddr){
	//Write operation (Send address and register to read)
	I2C1->CR2 = 0; //reset control reg2
	I2C1->CR2 |= ((uint32_t)devAddr << 1);//slave address
	I2C1->CR2 |= (1U << 16); //Number of bytes
	I2C1->CR2 |= (1U << 13); //Generate Start
	while(!(I2C1->ISR & (1U << 1))); //TXIS
	I2C1->TXDR = (uint32_t)regAddr;

	while(!(I2C1->ISR & (1U << 6))); //Transmission complete
```

```
      //Read operation (read data)
      I2C1->CR2 = 0;
      I2C1->CR2 |= ((uint32_t)devAddr << 1);
      I2C1->CR2 |= (1U << 10); //Read mode
      I2C1->CR2 |= (1U << 16); //Number of bytes
      I2C1->CR2 |= (1U << 15); //NACK=Not acknowledge
      I2C1->CR2 |= (1U << 25); //Autoend
      I2C1->CR2 |= (1U << 13); //Generate Start
      while(!(I2C1->ISR & (1U << 2)));//wait until RXNE=1

      uint8_t data = (uint8_t)I2C1->RXDR;
      return data;
}

void write_I2C(uint8_t devAddr, uint8_t regAddr, uint8_t data){
      //Write operation (Send address and register to read)
            I2C1->CR2 = 0;
            I2C1->CR2 |= ((uint32_t)devAddr << 1);//slave address
            I2C1->CR2 |= (2U << 16); //Number of bytes
            I2C1->CR2 |= (1U << 25); //AUTOEND
            I2C1->CR2 |= (1U << 13); //Generate Start
            while(!(I2C1->ISR & (1U << 1))); //TXIS
            I2C1->TXDR = (uint32_t)regAddr;
            while(!(I2C1->ISR & (1U << 1))); //TXIS
            I2C1->TXDR = (uint32_t)data;
}

void EEPROM_write_I2C(uint8_t devAddr,uint16_t memAddr, uint8_t* data,
int size){

      I2C1->CR2 = 0;
      I2C1->CR2 |= (uint32_t)(devAddr << 1);
      I2C1->CR2 |= (uint32_t)((size + 2)<< 16);
      I2C1->CR2 |= (1U << 25);      ///Auto-end/
      I2C1->CR2 |= (1U << 13);      ///Generate start/

      while(!(I2C1->ISR & (1 << 1)));     //high address
      I2C1->TXDR = (uint32_t)(memAddr >> 8);

      while(!(I2C1->ISR & (1 << 1)));     //low address
      I2C1->TXDR = (uint32_t)(memAddr & 0xFF);

      while(size){
            while(!(I2C1->ISR & (1 << 1)));
            I2C1->TXDR = (data++);  //DATA SEND
            size--;
      }
}

void EEPROM_read_I2C(uint8_t devAddr,uint16_t memAddr, uint8_t *data,
int size){
```

```
      I2C1->CR2 = 0;
      I2C1->CR2 |= (uint32_t)(devAddr << 1);
      I2C1->CR2 |= (2U << 16);      //Number of bytes
      I2C1->CR2 |= (1U << 13);      //Generate Start

      while(!(I2C1->ISR & (1 << 1)));//high address
      I2C1->TXDR = (uint32_t)(memAddr >> 8);

      while(!(I2C1->ISR & (1 << 1))); //low address
      I2C1->TXDR = (uint32_t)(memAddr & 0xFF);

      while(!(I2C1->ISR & (1 << 6)));     //is transmission complete

      //read data
      I2C1->CR2 = 0;
      I2C1->CR2 |= (uint32_t)(devAddr << 1);
      I2C1->CR2 |= (1U << 10);      //Read mode
      I2C1->CR2 |= (uint32_t)(size << 16);      //Number of bytes
      I2C1->CR2 |= (1U << 25);      //AUTOEND
      I2C1->CR2 |= (1U << 13);      //Generate start

      while(size){
            while(!(I2C1->ISR & (1 << 2)));
            (*data++) = (uint8_t)I2C1->RXDR;
            size--;
      }
}

void GPIO_Config(void){
  RCC->IOPENR |= (1U << 0); //Enable clock for GPIOA
  RCC->APBENR1 |= (1U << 17); //Enable clock for USART2
  GPIOA->MODER &= ~(3U << 2*2);
  GPIOA->MODER |= (2U << 2*2);
  GPIOA->AFR[0] &= ~(0xFU << 4*2);
  GPIOA->AFR[0] |= (1 << 4*2);
  GPIOA->MODER &= ~(0xFU << 2*3);
  GPIOA->MODER |= (2U << 2*3);
  GPIOA->AFR[0] &= ~(0xFU << 4*3);
  GPIOA->AFR[0] |= (1 << 4*3);
}

void print(char *buf){
  int len = 0;
  while(buf[len++] != '\0');
  _write(0, buf, len);
}

void printChar(uint8_t c){
  USART2->TDR = (uint16_t) c;
  while(!(USART2->ISR & (1 << 6))); // 6.bit transmission complete
}

int _write(int fd, char *ptr, int len) {
```

```
  (void)fd;
  for (int i=0; i<len; ++i){
    printChar(ptr[i]);
  }
  return len;
}

void USART_Config(uint16_t baud){
  USART2->CR1 = 0;
  USART2->CR1 |= (1U << 2); //USART1 receiver enable
  USART2->CR1 |= (1U << 3); //USART1 transmitter enable
  USART2->CR1 |= (1U << 5); //RX Interrupt enable
  USART2->BRR = (uint16_t)(SystemCoreClock / baud); //Setting
  USART2->CR1 |= (1U << 0); //USART2 enable
  NVIC_SetPriority(USART2_IRQn , 1);
  NVIC_EnableIRQ(USART2_IRQn);
}

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