GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRONICS ENGINEERING

ELEC 335

Microprocessors Laboratory

Lab #5 Experiment Report

Prepared by
200102002025 – Umut Mehmet ERDEM
200102002051 – Arda DERİCİ
200102002061 – Serdar BAŞYEMENİCİ

1. Introduction

The objective of ELEC 335 Laboratory #5 is to read, write and process analog values. To explain, a dimmer with a potentiometer will be applied. After connecting a potentiometer and two external LEDs that will light up in opposite configuration, the brightness of these LEDs will be changed by changing the potentiometer. Also, a knock counter will be implemented. After an SSD and microphone are connected to the circuit, when it is knocked on the table, the counter will increase one by one and this increase will be observed with the SSD.

2. Problems

2.1. Problem I

In this problem, a light dimmer with a potentiometer will be implemented.

2.1.1. Flow Chart and Schematic Diagram

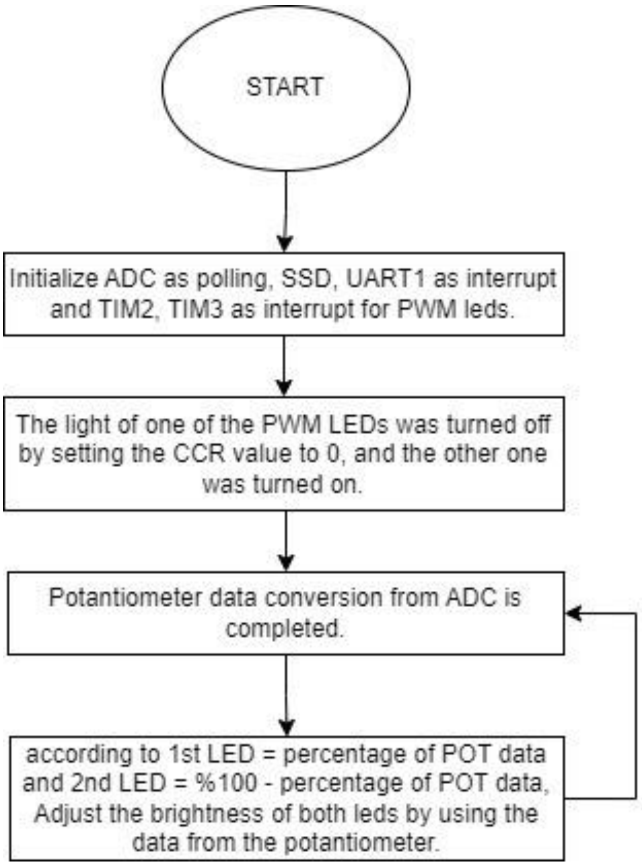

Figure 1 - Flow chart for Problem I.

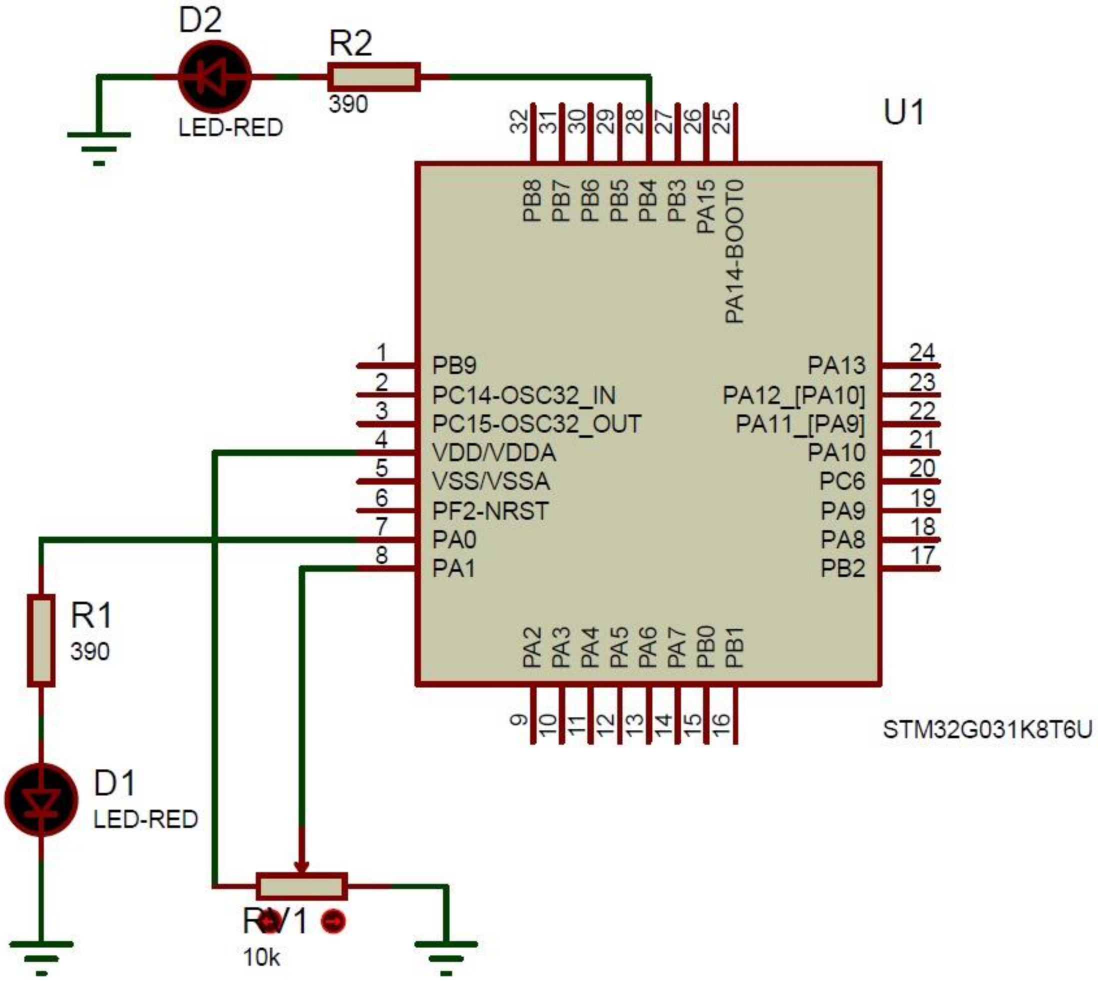

Figure 2 - Schematic diagram for Problem I.

2.1.2. Theoretical and Mathematical Work

$$DutyCycle_{PWM}[\%] = \frac{CCRx}{ARRx}[\%]$$

Figure 3 - Calculation of PWM.

$$F_{PWM} = \frac{F_{CLK}}{(ARR + 1) \times (PSC + 1)}$$

Figure 4 - Calculation of timer.

2.1.3. C Code of the Problem I

C code for Problem I is as follows:

```
/* author: Umut Mehmet ERDEM | Arda DERİCİ | Serdar BAŞYEMENİCİ
* problem1.c
*/

#include "stm32g0xx.h"

uint16_t pot_value; // Potantiometer data
uint32_t led1_val; // Brightness value for LED1
uint32_t led2_val; // Brightness value for LED2

uint16_t ADC_Data(void);
void GPIO_Config(void);
void TIM2_Config(void);
void TIM3_Config(void);
void ADC_Config(void);
void Delay(volatile uint32_t);

void TIM3_IRQHandler(void){
	TIM3->SR &=  ~(TIM_SR_CC1IF); // Clear capture compare interrupt 1 flag
	TIM3->SR &= ~(TIM_SR_UIF); // Clear update status register
}

void TIM2_IRQHandler(void) {
    // PWM Duty Cycle[%] = (CCRx/ARR)*100;
	pot_value = ADC_Data();
	led1_val = (16000*pot_value)/4095;
	led2_val = 16000-led1_val;

	TIM2->CCR1 =led1_val;
	TIM3->CCR1 =led2_val;

	TIM3->SR &=  ~(TIM_SR_CC1IF); // Clear capture compare interrupt 1 flag
    TIM2->SR &= ~(TIM_SR_UIF); // Clear update status register
}

int main(void){

	GPIO_Config();
	TIM2_Config();
	TIM3_Config();
	ADC_Config();

	while(1){

	}
	return 0;
}
```

```
uint16_t ADC_Data(void){
      ADC1->CR |= ADC_CR_ADSTART; // ADC Start Conversion

      /*This bit is set by hardware at the end of each conversion(EOC) of a
channel
       * when a new data result is available in the ADC_DR register.
       * 0: Channel conversion not complete
       * 1: Channel conversion complete*/
      if((ADC1->ISR>>2) & (ADC_ISR_EOC>>2)){
            return ADC1->DR; // return ADC data value
      }
      return 0;
}

void GPIO_Config(void){
      // input-output A and B ports clock enable
      RCC->IOPENR |= (RCC_IOPENR_GPIOAEN | RCC_IOPENR_GPIOBEN);

      // select PA0 mode as Alternate Function
      GPIOA->MODER &= ~(3U << 2*0);
      GPIOA->MODER |= (2U << 2*0);

      /* PA0 pin used for TIM2_CH1 are selected
       * with GPIOx_AFRL = AFRL_AFSELy(Alternate Function register -
       * Alternate function selection for port x pin y)
       * AF2 -->> TIM2_CH1*/
      GPIOA->AFR[0] |= GPIO_AFRL_AFSEL0_1;

      // select PB4 mode as Alternate Function
      GPIOB->MODER &= ~(3U << 2*4);
      GPIOB->MODER |=  (2<< 2*4) ;

      /* PB4 pin used for TIM3_CH1 are selected
      * AF1 -->> TIM3_CH1*/
      GPIOB->AFR[0]  |= GPIO_AFRL_AFSEL4_0;

      //PA1 is ADC
      GPIOA->MODER |= (3 << 2*1);
}

void ADC_Config(void){
      RCC->APBENR2 |= RCC_APBENR2_ADCEN; // ADC clock enable

      ADC1->CR |=ADC_CR_ADVREGEN; //voltage regulator enable
      Delay(500);

      ADC1->CR |=ADC_CR_ADCAL; //calibration
      while(((ADC1->CR>>31)==ADC_CR_ADCAL>>31)); // until calibration
      /* 0: Calibration complete
       * 1: Write 1 to calibrate the ADC. Read at 1 means that a calibration
       * is in progress.*/
```

```
      ADC1->CR |= ADC_CR_ADEN; // ADC is enabled.
      while (ADC1->ISR & ADC_ISR_ADRDY); // 1: ADC is ready to start
conversion

      ADC1->CHSELR |= ADC_CHSELR_CHSEL1; //chanel selection for PA1
      ADC1->CFGR1 |= ADC_CFGR1_CONT; // contionous conversion

      ADC1->SMPR |= (6UL<<0); // 79.5 ADC clock cycles for sampling time
selection 1
}

void TIM2_Config(void){
      RCC->APBENR1 |= RCC_APBENR1_TIM2EN; // Timer 2 clock enable

      TIM2->CR1 = 0; // zero out the control register just in case
      TIM2->CR1 |= TIM_CR1_ARPE; // Auto-reload preload enable

      TIM2->CCMR1 |= (6U << 4); // PWM mode 1 is selected.
      TIM2->CCMR1 |= TIM_CCMR1_OC1PE; // Output Compare 1 Preload Enable

      TIM2->CCER |= TIM_CCER_CC1E; // Capture compare ch1 enable

      TIM2->CNT = 0; // zero out counter

      // tim update frequency = TIM_CLK/((TIM_PSC+1)*TIM_ARR) for 1s interrupt
      TIM2->PSC = 9; // prescaler
      TIM2->ARR = 16000; // period

      TIM2->CCR1 = 0; // zero out duty for ch1 in TIM capture/compare register
1

      // Update Generation: Re-initialize the counter and generates an update
of the registers.
      TIM2->EGR |= TIM_EGR_UG;

      TIM2->DIER |= TIM_DIER_UIE; // Update interrupt enable

      TIM2->CR1 |= TIM_CR1_CEN; // TIM2 Counter enable

      NVIC_SetPriority(TIM2_IRQn, 1); // Setting Priority for timer handler
      NVIC_EnableIRQ(TIM2_IRQn); // timer handler enable
}

void TIM3_Config(void){
      RCC->APBENR1 |=  RCC_APBENR1_TIM3EN; // Timer 3 clock enable

      TIM3->CR1=0; // zero out the control register just in case
      TIM3->CR1 |= TIM_CR1_ARPE; // Auto-reload preload enable

      TIM3->CCMR1 |= (6U << 4); // PWM mode 1 is selected. -->> 0110: PWM Mode
      TIM3->CCMR1 |= TIM_CCMR1_OC1PE; // Output Compare 1 Preload Enable
```

```
	TIM3->CCER |= TIM_CCER_CC1E; // Capture compare ch1 enable

	TIM3->CNT =0; // zero out counter

	// tim update frequency = TIM_CLK/((TIM_PSC+1)*TIM_ARR) for 1s interrupt
	TIM3->PSC= 9;
	TIM3->ARR= 16000;

	// Update Generation: Re-initialize the counter and generates an update
of the registers.
	TIM3->EGR |= TIM_EGR_UG;

	TIM3->DIER |= TIM_DIER_UIE; // Update interrupt enable

	TIM3->CR1 |= TIM_CR1_CEN; // TIM3 Counter enable

	NVIC_SetPriority(TIM3_IRQn, 1);
	NVIC_EnableIRQ(TIM3_IRQn);
}

void Delay(volatile uint32_t time){
    for(; time>0; time--);
}
```

First of all, reset and clock control are activated by assigning them to registers with RCC_APBENR1_TIM2EN, RCC_APBENR1_TIM3EN, RCC_APBENR2_ADCEN, RCC_IOPENR_GPIOAEN, RCC_IOPENR_GPIOBEN for each peripheral unit to be used. In the TIM2 and TIM3 configuration functions, the PWM mode and interrupt of the pins defined as alternative functions in the GPIO configuration function are activated; PSC, ARR, CCR values are assigned. Similarly, in the ADC configuration function, ADC is activated and the number of sampling cycles and continuous cycles for ADC are set. In the TIM2 cutting function, the potentiometer data is received with the ADC Data Register in the ADC_Data function, and the led1 and led2 values is assigned according to the formula %led2 = %led1 - %potentiometer value, with a total percentage value of 100%.

2.2. Problem II

In this problem, a knock counter will be implemented.

2.2.1. Flow Chart and Schematic Diagram

The flow chart and schematic diagram for Problem 2 are in Figure 5 and Figure 6.

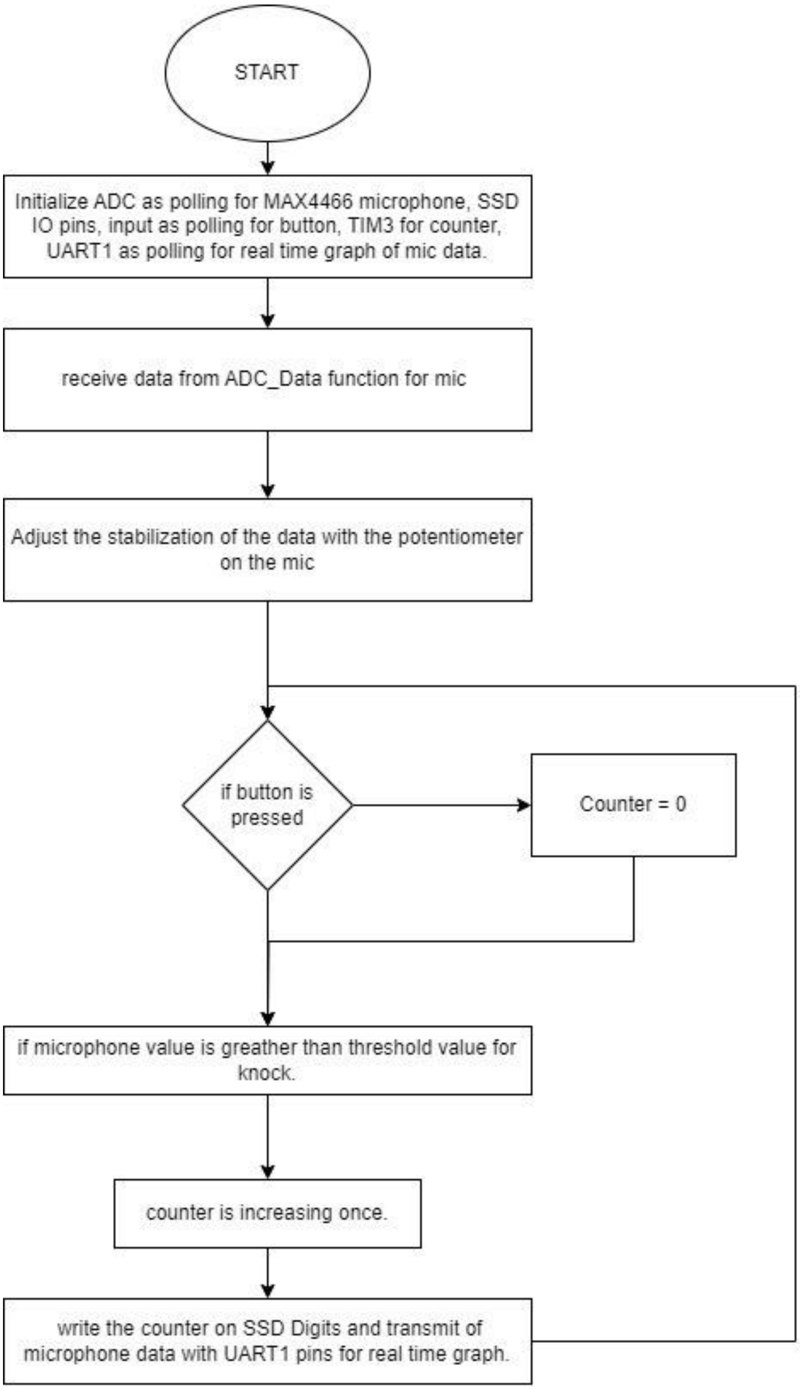

Figure 5 - Flow chart for Problem II.

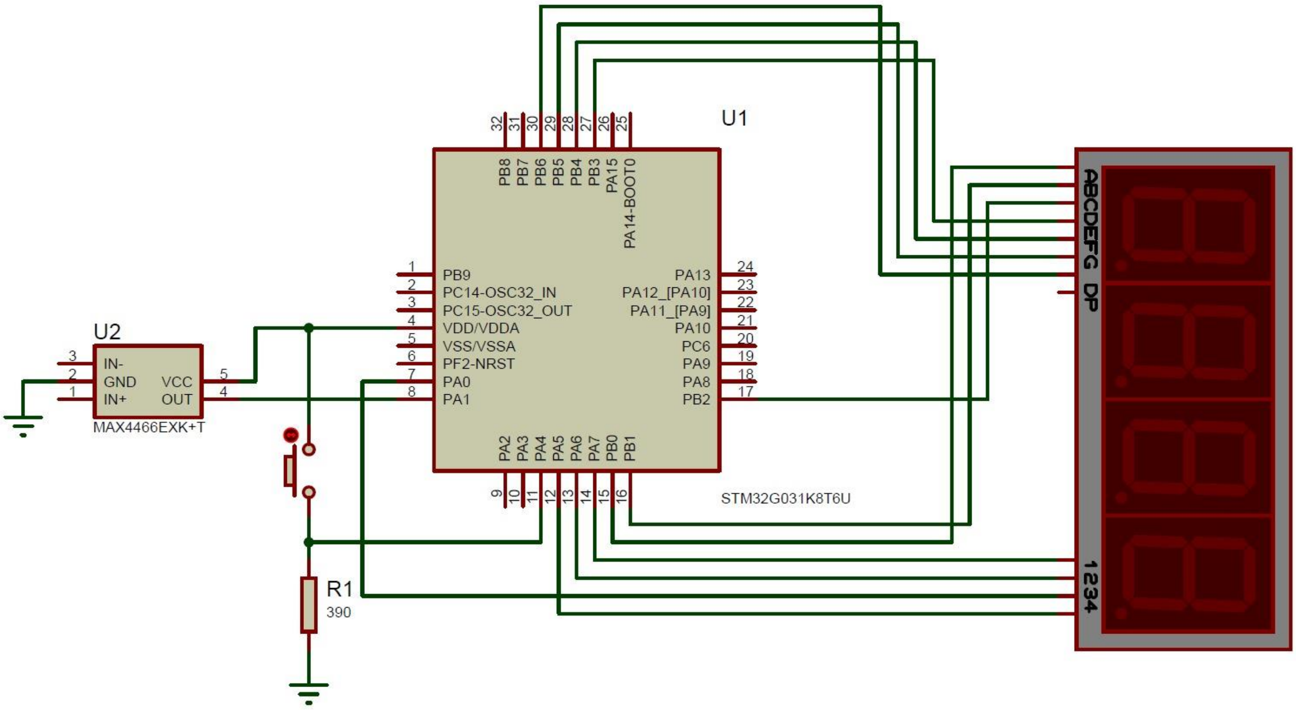

Figure 6 – Schematic diagram for Problem II.

2.2.2. Theoretical and Mathematical Work

Peripheral Clock

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\mathbf{8} \times \text{USARTDIV}} \quad \text{if OVER8} = 1$$

Divide factor to generate different baud rates

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\mathbf{16} \times \text{USARTDIV}} \quad \text{if OVER8} = 0$$

Figure 7 - Calculation of 9600 baud rate.

$$F_{\text{Timer}} = \frac{F_{CLK}}{(ARR + 1) \times (PSC + 1)}$$

Figure 8 - Calculation of timer.

2.2.3. C Code of the Problem II

C code for Problem II is as follows;

```
/* author: Umut Mehmet ERDEM | Arda DERİCİ | Serdar BAŞYEMENİCİ
* problem2.c
*/
#include "stdlib.h"
#include "stm32g0xx.h"

uint16_t mic_val; // microphone data
uint16_t counter = 0;
uint16_t ADC_Data(void);
void System_Config(void);
void GPIO_Config(void);
void TIM3_Config(void);
void USART1_Config(void);
void ADC_Config(void);
void counterDisplay(uint16_t);
void printInt(uint16_t);
void clearSSD(void);
void setSSD(int);
void SetZero(void);
void print(char *);
int _print(int, char *, int);
void printChar(uint8_t);
void Delay(volatile uint32_t);

void TIM3_IRQHandler(void){
	mic_val = ADC_Data();

	if((GPIOA->IDR >>4) & 1){ // if button is pressed
		counter = 0;
		Delay(32000);
	}
	if(mic_val > 96){
		counter++;
		Delay(320000);
	}
	counterDisplay(counter);
	Delay(2000);

	TIM3->SR &=  ~(TIM_SR_CC1IF); // Clear capture compare interrupt 1 flag
	TIM3->SR &= ~(TIM_SR_UIF); // Clear update status register
}

int main(void){
	System_Config();
	while(1){
		printInt(mic_val); // transmit of microphone data for real time
graph
	}
	return 0;
}
```

```
void System_Config(void){// all initialize are this function
      SetZero(); // leds show us 0000 value.
      GPIO_Config();
      TIM3_Config();
      USART1_Config();
      ADC_Config();
}

void counterDisplay(uint16_t counterVal){
      int thousand, hundred, decimal, unit;
      thousand=(counterVal/1000); // thousand digit of counter
      hundred=((counterVal-thousand*1000)/100); // hundred digit of counter
      decimal=((counterVal- thousand*1000 - hundred*100)/10); // decimal digit
of counter
      unit=(counterVal- thousand*1000 - hundred*100 - decimal*10); // unit
digit of counter

      /* unit digit we want is set to 1 and the others are set to 0*/
      GPIOA ->ODR &= ~(1U << 7); // off D1 - PA7
      GPIOA ->ODR &= ~(1U << 6); // off D2 - PA6
      GPIOA ->ODR &= ~(1U << 0); // off D3 - PA0
      GPIOA ->ODR |= (1U << 5); // on D4 - PA5
      setSSD(unit);
      Delay(100);

      /* decimal digit we want is set to 1 and the others are set to 0*/
      GPIOA ->ODR &= ~(1U << 7); // D1 - PA7
      GPIOA ->ODR &= ~(1U << 6); // D2 - PA7
      GPIOA ->ODR |= (1U << 0); // D3 - PA7
      GPIOA ->ODR &= ~(1U << 5);
      setSSD(decimal);
      Delay(100);

      /* hundred digit we want is set to 1 and the others are set to 0*/
      GPIOA ->ODR &= ~(1U << 7); // D1 - PA7
      GPIOA ->ODR |= (1U << 6); // D2 - PA7
      GPIOA ->ODR &= ~(1U << 0); // D3 - PA7
      GPIOA ->ODR &= ~(1U << 5);
      setSSD(hundred);
      Delay(100);

      /* thousand digit we want is set to 1 and the others are set to 0*/
      GPIOA ->ODR |= (1U << 7); // D1 - PA7
      GPIOA ->ODR &= ~(1U << 6); // D2 - PA7
      GPIOA ->ODR &= ~(1U << 0); // D3 - PA7
      GPIOA ->ODR &= ~(1U << 5);
      setSSD(thousand);
      Delay(300);
}
```

```
uint16_t ADC_Data(void){
	ADC1->CR |= ADC_CR_ADSTART; // ADC Start Conversion

	/*This bit is set by hardware at the end of each conversion(EOC) of a
channel
	 * when a new data result is available in the ADC_DR register.
	 * 0: Channel conversion not complete
	 * 1: Channel conversion complete*/
	if((ADC1->ISR>>2) & (ADC_ISR_EOC>>2)){
		return ADC1->DR; // return ADC data value
	}
	return 0;
}

void printInt(uint16_t intVal){ // convert integer to alphabet
	char buffer[5];
	snprintf(buffer, 5, "%d\n\r", intVal);
	print(buffer);
}
void print(char *s){
	int length = 0; // to count length of character
	/* i is pointer of string and length is increasing until i equals NULL
character*/
	for(char *i = s; *i != NULL; i++) length++;
	_print(0, s, length);
}

int _print(int f, char *ptr, int len){
	/*in for loop, i of is increasing until equal to len
	 * and meanwhile, chars of 2nd parameter of _print function is writen
	 * into the printChar character by character increasing ptr of 2nd
parameter
	 * of _print function  */
	for(volatile int i = f; i<len; i++){
		printChar(*ptr);
		ptr++;
	}
	return len; // return length
}

void printChar(uint8_t c){
	while(!(USART1->ISR & USART_ISR_TXE_TXFNF)); // when messages are sent.
	USART1->TDR = c; // Transmit data register is taken character to send a
message.
}

void GPIO_Config(void){
	// input-output A and B ports clock enable
	RCC->IOPENR |= (RCC_IOPENR_GPIOAEN | RCC_IOPENR_GPIOBEN);

	// PA4 is set as input for button
	GPIOA->MODER &= ~(3U << 2*4);
```

```
	//PA1 is ADC for microphone
	GPIOA->MODER |= (3 << 2*1);

	/* modes of GPIOA PA9 and PA10 pins are selected as alternate function.
	 * like that 0b1111_1010_1111;*/
	GPIOA->MODER &= ~((3U << 2*9) | (3U << 2*10));
	GPIOA->MODER |= (2U << 2*9) | (2U << 2*10);

	/* PA9 and PA10 pins used for USART1_TX and USART1_RX are selected
	 * with GPIOx_AFRH = AFRH_AFSELy(Alternate Function register -
	 * Alternate function selection for port x pin y)
	 * AF1 -->> USART1_RX, USART1_TX*/
	GPIOA->AFR[1] |= GPIO_AFRH_AFSEL9_0;
	GPIOA->AFR[1] |= GPIO_AFRH_AFSEL10_0;

	/* enable required GPIOA registers and RCC register */
	/*PA7 -> D1 digit, PA6 -> D2 digit, PA0 -> D3 digit, PA5 -> D4 digit,*/
	RCC->IOPENR |= (1U << 0);
	for(int k=0; k<9; k++){
		if (k==0 || k==1 || k==5 || k==6 || k==7 || k==8){
			GPIOA->MODER &= ~(3U << 2*k);
			GPIOA->MODER |= (1U << 2*k);
		}
	}

	/* enable required GPIOB registers and RCC register */
	/*PB0-PB6 output pins are assigned from A to G respectively*/
	RCC->IOPENR |= (1U << 1);
	for(int k=0; k<9; k++){
		if (k==0 || k==1 || k==2 || k==3 || k==4 || k==5 || k==6 || k==8){
			GPIOB->MODER &= ~(3U << 2*k);
			GPIOB->MODER |= (1U << 2*k);
		}
	}
}

void ADC_Config(void){
	RCC->APBENR2 |= RCC_APBENR2_ADCEN; // ADC clock enable

	ADC1->CR |=ADC_CR_ADVREGEN; //voltage regulator enable
	Delay(500);

	ADC1->CR |=ADC_CR_ADCAL; //calibration
	while(((ADC1->CR>>31)==ADC_CR_ADCAL>>31)); // until calibration
	/* 0: Calibration complete
	 * 1: Write 1 to calibrate the ADC. Read at 1 means that a calibration
	 * is in progress.*/

	ADC1->CR |= ADC_CR_ADEN; // ADC is enabled.
	while (ADC1->ISR & ADC_ISR_ADRDY); //1: ADC is ready to start conversion
```

```
      ADC1->CHSELR |= ADC_CHSELR_CHSEL1; //chanel selection for PA1
      ADC1->CFGR1 |= ADC_CFGR1_CONT; // contionous conversion

      ADC1->SMPR |= (6UL<<0); // 79.5 ADC clock cycles for sampling time
selection 1
}

void TIM3_Config(void){
      RCC->APBENR1 |=   RCC_APBENR1_TIM3EN; // Timer 3 clock enable

      TIM3->CR1=0; // zero out the control register just in case
      TIM3->CR1 |= TIM_CR1_ARPE; // Auto-reload preload enable

      TIM3->CNT =0; // zero out counter

      // tim uptade frequency = TIM_CLK/((TIM_PSC+1)*TIM_ARR) for 1s interrupt
      TIM3->PSC= 0;
      TIM3->ARR= 16000;

      TIM3->DIER |= TIM_DIER_UIE; // Update interrupt enable

      TIM3->CR1 |= TIM_CR1_CEN; // TIM3 Counter enable

      NVIC_SetPriority(TIM3_IRQn, 1);
      NVIC_EnableIRQ(TIM3_IRQn);
}

void USART1_Config(void){
      RCC->APBENR2 |= RCC_APBENR2_USART1EN; // RCC APB peripherals clock
enable for USART2
      USART1->CR1 = 0x00; // clear all
      USART1->CR1 |= USART_CR1_UE; // UE: USART enable

      /* Baud rate of 9600, PCLK1 at 16 MHz
       * TX/RX baud rate = f_clk/(16*USARTDIV)
       * 9600 = 16MHz/(16*USARTDIV) --->>> USARTDIV = 104.1666667
       * IEEE754 floating-point --->>> mantissa = 104, fraction = 0.167*16 =
2.672 ≈ 3*/
      USART1->BRR |= (3 << 0) | (104 << 4);

      USART1->CR1 |= USART_CR1_RE; // RE: Receiver enable
      USART1->CR1 |= USART_CR1_TE; // TE: Transmitter enable
}

void clearSSD(void){ // Clear display
      GPIOB -> ODR |= (1U << 0); //PB0 -> A
      GPIOB -> ODR |= (1U << 1); //PB1 -> B
      GPIOB -> ODR |= (1U << 2); //PB2 -> C
      GPIOB -> ODR |= (1U << 3); //PB3 -> D
      GPIOB -> ODR |= (1U << 4); //PB4 -> E
      GPIOB -> ODR |= (1U << 5); //PB5 -> F
      GPIOB -> ODR |= (1U << 6); //PB6 -> G
}
```

```
void setSSD(int x){ // choose number we want and its leds are turned on.
	clearSSD();
	switch(x){
		case 0:
			GPIOB->ODR &= ~(0x3F); // A,B,C,D,E,F is on
			break;
		case 1:
			GPIOB->ODR &= ~(0x6); // B,C is on
			break;
		case 2:
			GPIOB->ODR &= ~(0x5B); // A,B,D,E,G is on
			break;
		case 3:
			GPIOB->ODR &= ~(0x4F); // A,B,C,D,G is on
			break;
		case 4:
			GPIOB->ODR &= ~(0x66); // B,C,F,G is on
			break;
		case 5:
			GPIOB->ODR &= ~(0x6D); // A,C,D,F,G is on
			break;
		case 6:
			GPIOB->ODR &= ~(0x7D); // A,C,D,E,F,G is on
			break;
		case 7:
			GPIOB->ODR &= ~(0x7); // A,B,C is on
			break;
		case 8:
			GPIOB->ODR &= ~(0x7F); // A,B,C,D,E,F,G is on
			break;
		case 9:
			GPIOB->ODR &= ~(0x6F); //A,B,C,D,F,G is on; E is off
			break;
	}
}

void SetZero(void){
	GPIOA ->ODR |= (1U << 7); // D1 digit -> PA7
	GPIOA ->ODR |= (1U << 6); // D2 digit -> PA6
	GPIOA ->ODR |= (1U << 0); // D3 digit -> PA0
	GPIOA ->ODR |= (1U << 5); // D4 digit -> PA5
	setSSD(0);
}
void Delay(volatile uint32_t time){
    for(; time>0; time--);
}
```

First of all, reset and clock control are activated by assigning them to registers with RCC_APBENR1_TIM3EN, RCC_APBENR2_ADCEN, RCC_APBENR2_USART1EN, RCC_IOPENR_GPIOAEN, RCC_IOPENR_GPIOBEN for each peripheral unit to be used. The interrupt of the counter defined as an alternative function in the GPIO configuration function is activated in the TIM3 configuration function; PSC, ARR values are assigned. Similarly, in the ADC configuration function, ADC are activated and the number of sampling cycles and continuous cycles for ADC are set. Pins are designated as alternative functions in the GPIO configuration function for UART1; uart, receiver and transmitter are activated by assigning the baud rate value to the Baud rate register (BRR). For Seven Segment Display, all digits are reset to zero in the SetZero function, and in the setSSD function, pins are determined to be assigned values between 0-9 for each digit. Within the counterDisplay function, for the number to be sent to the Seven Segment Display, the digits flash and light quickly, respectively, according to the place value, and the values appear on the display. The microphone value for UART1 is sent by converting it from number to character with the printInt function, and this value is graphed in real time. These operations are respectively in the TIM3 interrupt function:

- reading microphone value with ADC_Data function
- checking whether the button was pressed and resetting the counter if it was pressed
- checking if mic_val is greater than 96 and incrementing the counter by 1 if greater
- showing mic_val value on SSD with counterDisplay function

In the while loop within the main function, the mic_val value is sent via uart.

Also, Python code written to show the graph of the microphone in real time is as follows.

```
import serial
import matplotlib.pyplot as plt
from drawnow import drawnow

# Seri port ayarları
ser = serial.Serial('COM6', 9600)  # 'COMx' kısmını kullanmak istediğin port
ile değiştir

# Grafik için boş liste
data = []

# Grafik güncelleme fonksiyonu
def update_graph():
    plt.plot(data, label='UART Verisi')
    plt.xlabel('Zaman')
    plt.ylabel('Veri Değeri')
    plt.title('Real-Time Grafik')
    plt.legend()

# Ana döngü
while True:
    # UART verilerini oku
    print(ser.readline().decode('utf-8'))
    uart_data = ser.readline().decode('utf-8').strip()
```

```
    # Veriyi işle ve listeye ekle
    try:
        data_point = uart_data
        #print(uart_data)
        data.append(data_point)
    except ValueError:
        print("Hatalı veri formatı:", uart_data)
        continue

    # Grafik güncelle
    drawnow(update_graph)

    # Grafikte son 50 veriyi tut
    if len(data) > 20:
        data.pop(0)
```

The output of the Python code block above can be seen in Figure 9.

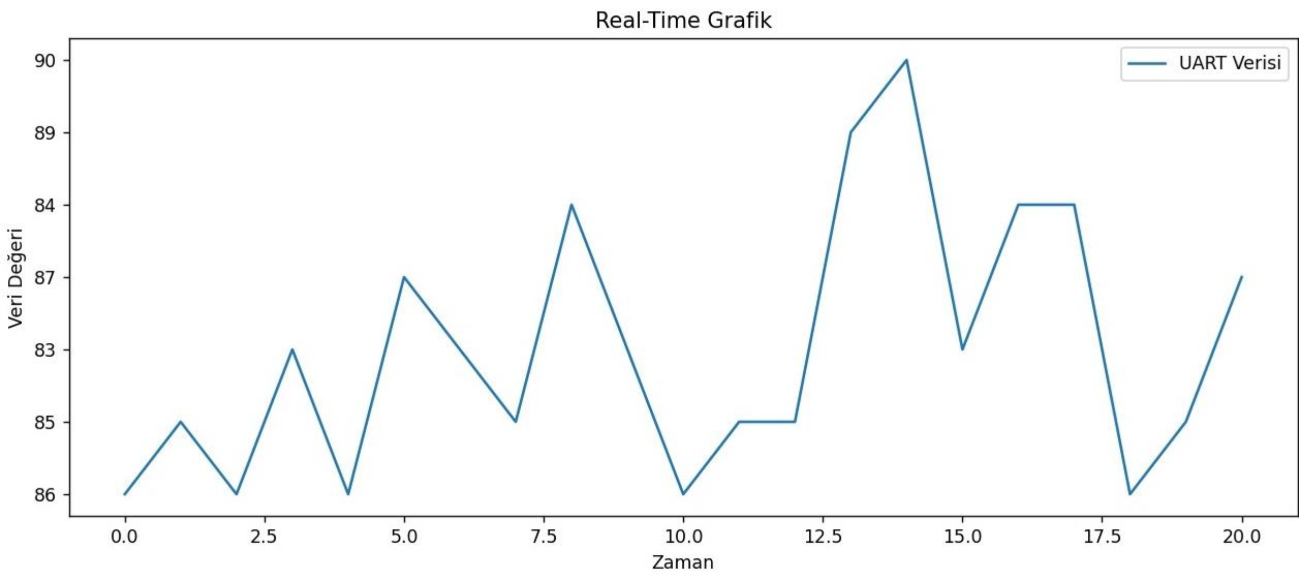

Figure 9 - Graph of the microphone at any given moment.

3. Conclusions and Comments

In Lab #5, the focus is on interfacing with analog sensors, specifically a microphone and a potentiometer, using the Analog-to-Digital Converter (ADC). The primary objective is to learn how to read analog signals and utilize the acquired values in a sample application. This hands-on experience are allowed for gaining insights into the accuracy of sensor readings, which are reflected in the control of LEDs and Seven-Segment Displays (SSDs). The ADC working principle is a key aspect of the lab, highlighting the conversion of continuous analog signals into discrete digital values. By connecting a microphone and a potentiometer to the ADC, it is could capture real-world variations in sound intensity and resistance. The practical application is involved using the acquired sensor values to control LEDs and SSDs, showcasing the potential of ADC data in creating responsive and interactive systems. Moreover, lab #5 is extended its exploration into the realm of data visualization by incorporating Python code to draw real-time graphs of the microphone's values. In summary, lab #5 is provided a holistic learning experience by combining theoretical knowledge of ADC principles with hands-on applications involving analog sensors.

4. References

- https://github.com/fcayci/stm32g0
- https://www.st.com/resource/en/reference_manual/rm0444-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
- https://www.st.com/resource/en/datasheet/stm32g031k8.pdf
- https://www.st.com/resource/en/schematic_pack/mb1455-g031k8-c01_schematic.pdf
- https://www.st.com/resource/en/user_manual/um2591-stm32g0-nucleo32-board-mb1455-stmicroelectronics.pdf
- drawio.com