1. INRODUCTION:

The objective of this lab is to is to read, write and process analog values. Used C language for the problems unless some parts require inline assembly. Used blinky project from stm32g0 repo as the starting point for your problems.

2. PROBLEMS:

2.1. Problem 1:

In this problem, you will implement a light dimmer with a potentiometer.

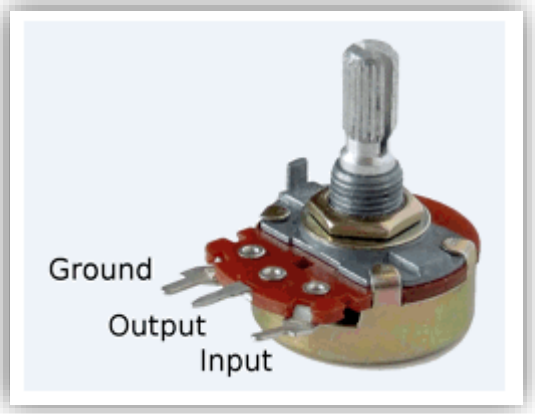
- Connect a pot using a resistor divider setting.
- Connect two external LEDs. These LEDs will light up in opposite configuration.
- By changing the pot you will change the brightness of these LEDs. For example if the pot is all they way down, first LED should light up, and second LED should be o, and if the pot is all the way up, first LED should be o and the second LED should light up. Their brightness should change in between.
- You will need PWM for the LED driving to change the brightness. 0 duty cycle will turn o the LEDs and 100% duty cycle will light them up completely

2.1.1. Theoretical Research:

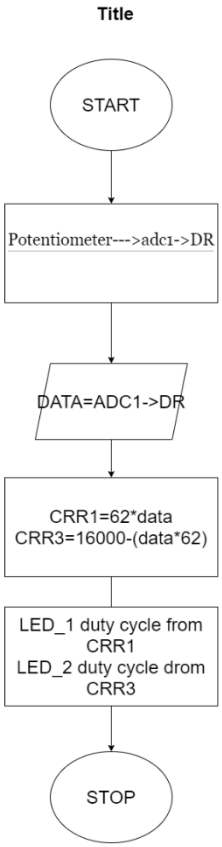
Potentiometer:

A **potentiometer** is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a **variable resistor** or **rheostat**.

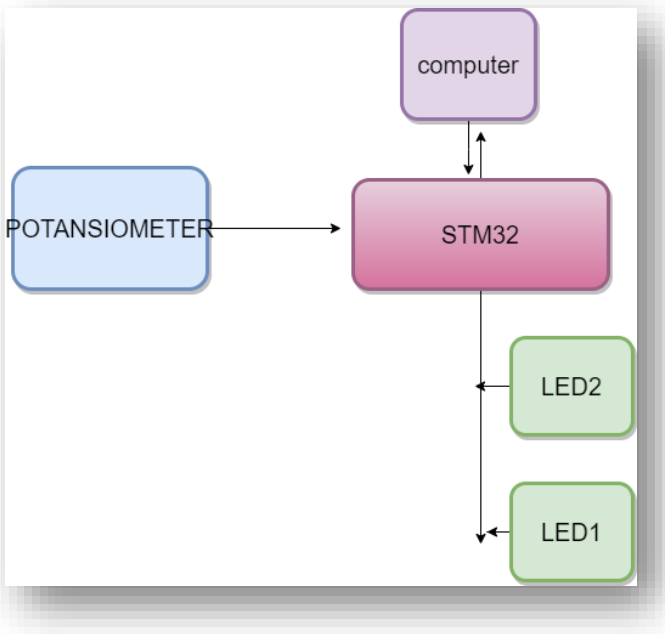
The measuring instrument called a potentiometer is essentially a voltage divider used for measuring electric potential (voltage); the component is an implementation of the same principle, hence its name.



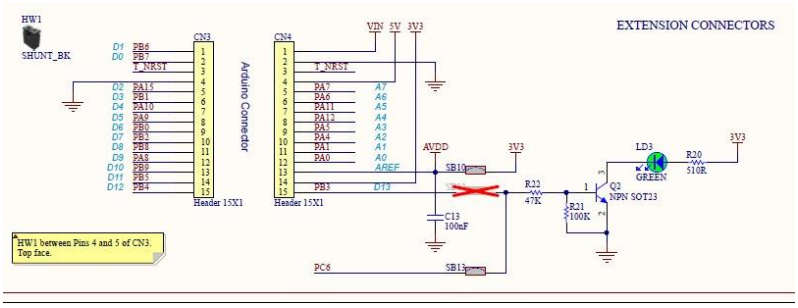
2.1.2 Flowchart:



2.1.3 Block diagram:



2.1.4 Schematic:



2.1.5 Code: Added on Appendix-A

2.2 PROBLEM-2

In this problem, you will work on implementing a knock counter.

- Connect SSD that will show the number of knocks.
- Connect an external button that will reset the counter.
- Connect a microphone that will pick up the sounds.
- When you knock on the table, you should increment the counter by one.
- There should be no mis-increments, or multiple increments as much as possible.

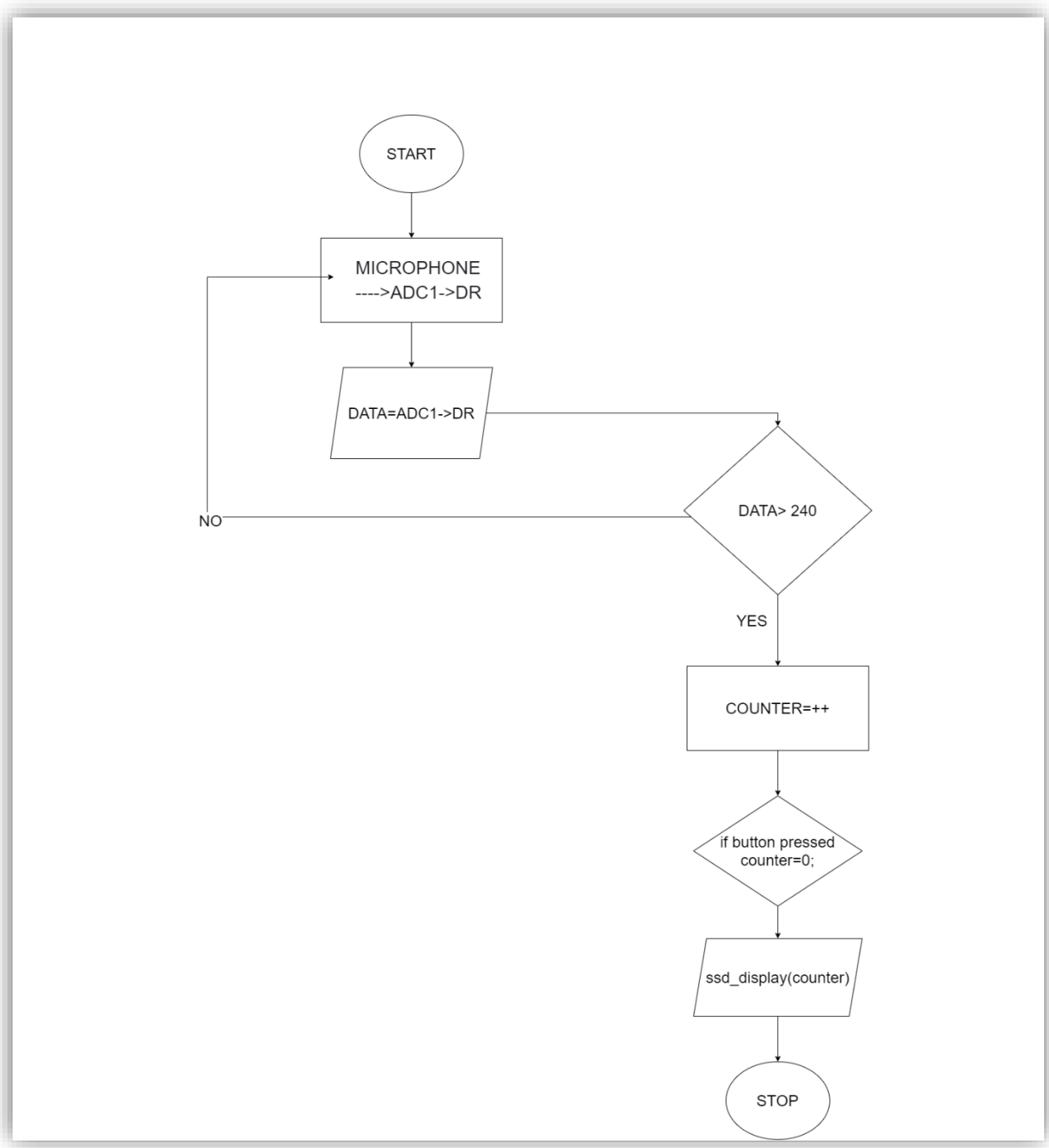
2.2.1 Theoretical Research:

Microphone:

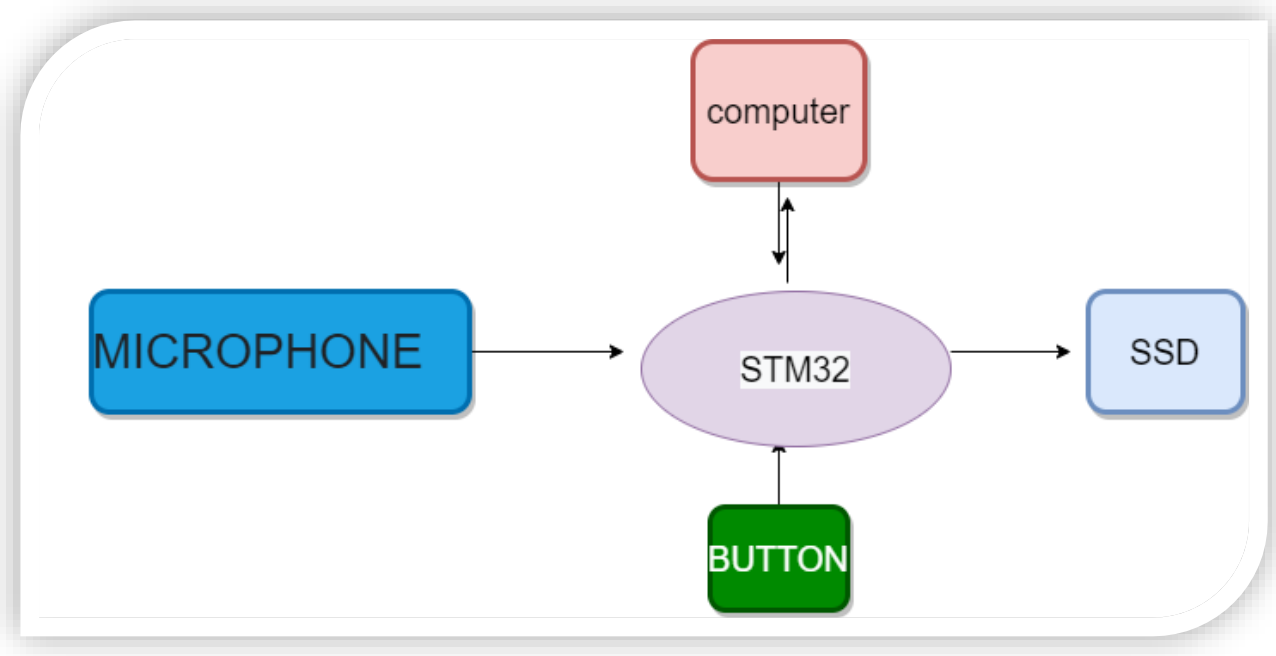
A microphone is a device that captures audio by converting sound waves into an electrical signal. This signal can be amplified as an analog signal or may be converted to a digital signal, which can be processed by a computer or other digital audio device.

While all microphones (or "mics") serve the same basic function, they can capture audio in several different ways. Therefore, multiple classes of microphones exist.

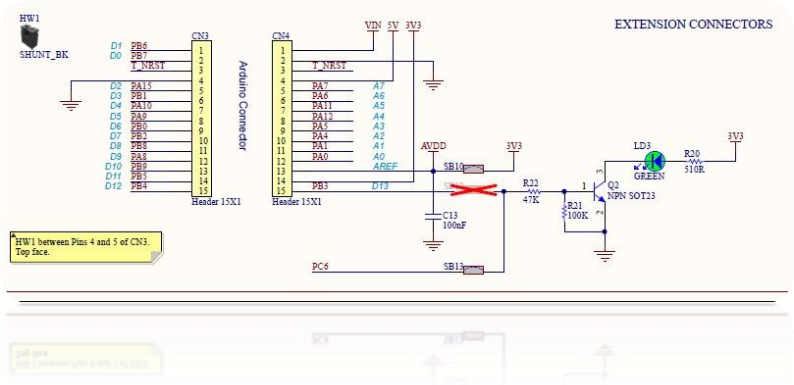
2.2.2 Flowchart:



2.2.3 Block diagram:



2.2.4 Schematic:



2.2.2 Code: Added on Appendix-B

3. CONCLUSION

The purpose of this experiment is read, write and process analog values C code. In Problem 1, we connect a potentiometer and 2 LEDs to the circuit requested from us. Then, when turning the potentiometer, the brightness of the LEDs was asked to change between 0 duty cycle and 100 duty cycle. First, we read the data from the potentiometer using ADC for this problem.

The steps for the ADC are as follows:

- Choose the relevant GPIO pin as Analog Input (Table 12 from stm32g031k8 datasheet)
- Enable ADC Clock
- Enable ADC Voltage Regulator, wait for at least 20us
- Enable Calibration, wait until completion
- Enable end of calibration or sequence interrupts
- 8 Configured number of bits to sample
- Configure single
- Select sampling time from SMPR register
- Enable channels (for the AN5 pins) we used PA5
- Enable ADC and wait until it is ready
- Start conversion

Then we created 2 different pwm with timer3. One of the pins is PB0 and the other is PA6. Signals for CH1 and CH3 were generated for these pins. CRR1 and CRR3 registers were used to determine the duty cycle of the PWM signals. Data between 0-255 coming from the potentiometer were assigned to these registers after multiplying by a factor of 62. Thus, the values coming from the potentiometer and the duty cycles of the leds were adjusted.

For Problem 2:

In this problem we were asked to make a knock counter. We were asked to count these knockes with the microphone and collect them using a counter. And we were asked to display this counter on SSD. And finally, it was asked to connect a button and reset the counter of this button. In this problem, the ADC used in problem 1 was used. Two different timers were used, one to keep the SSD on and the other to collect data from the microphone with the ADC. Since the microphone is very sensitive, only a certain decibel range is used by filtering the incoming data. If the microphone captures a sound at a certain height, it assigned this sound to the data with one the ADC-> DR register. This incoming value has increased the counter and the increasing counter is digitized with the utility_SSD () function. Then it was displayed on the SSD with the printDigit_SSD () and display_SSD () functions. When the button is pressed, counter is reset and 0000 is printed on the screen. Problems have been successfully completed. The difficulties encountered are; the microphone is sensitive and the generated sound lasts more than 1 cycle, so the counter sometimes counts more than once.

-APPENDIX A-

ADC.h

```c
/*
 * ADC.h
 *
 *  Created on: 5 Oca 2021
 *      Author: Mehmet Akif Gümüş
 */

#ifndef ADC_H_
#define ADC_H_

#include "stm32g0xx.h"

void init_ADC();

#endif /* ADC_H_ */

```

ADC.c

```c
/*
 * ADC.c
 *
 *  Created on: 5 Oca 2021
 *      Author: mehme
 */

#include "ADC.h"


void init_ADC(void){

    RCC->IOPENR |= (1U << 0);
    RCC->APBENR2 |= (1U << 20);// enable ADC clock

    //setup PA5 as analog
    GPIOA->MODER &= ~(3U << 2*5);
    GPIOA->MODER |= (3U << 2*5);

    ADC1->CR |= (1U << 28); //ADC voltage regulator enabled

    for(uint32_t i=0; i> 0xFFFF; i++);

    ADC1->CR |= (1U << 31);//ADC calibration enabled

    while(0 != (ADC1->CR & (1U << 31)));//wait until completion

    ADC1->IER |= (1U << 2);//End of conversion interrupt enable

    ADC1->CFGR1 |= (2U << 3);// 10: 8 bits

    /* single conversion mode*/

    ADC1->SMPR |= (4U << 0);//Sampling time selection 1 as 100: 19.5 ADC clock cycles


    ADC1->CHSELR |= (1U << 5);//1: Input Channel-5 is selected for conversion

    ADC1->CR |= (1U << 0); //ADC enable command

    while(0 == (ADC1->ISR & (1U << 0)));// 1: ADC is ready to start conversion


}

```

bsp.h

```h
#ifndef BSP_H_
#define BSP_H_

#include "stm32g0xx.h"
#include "ADC.h"

void BSP_system_init();

void init_timer2();

#endif

```

bsp.c

```c
#include "bsp.h"

uint32_t data;
void BSP_system_init(void){

    __disable_irq();

        init_ADC();
        init_timer2();
    __enable_irq();


}

void init_timer2(){

    RCC->IOPENR |= 7;
    RCC->APBENR1 |= (1U<< 1);// 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 << 4*6);
    GPIOA->AFR[0] |= (1U << 4*6);

    //setup PB0 as AF1
    GPIOB->MODER &= ~(3U << 0);
    GPIOB->MODER |= (2U << 0);

    // choose AF1 from mux
    GPIOB->AFR[0] &= ~(0xFU << 0);
    GPIOB->AFR[0] |= (1U << 0);



    TIM3->CR1=0;// zero out the control register just in case
    TIM3->CR1 |= (1U << 7); // ARPE
    TIM3->CNT=0;// zero out counter

    /*10 Msecond interrupt   */

    TIM3->PSC = 10;
    TIM3->ARR = 16000;
    TIM3->DIER |= (1 << 0);// update interrupt enable


    //PWM FOR PA6 TIM3_CH1
    TIM3->CCMR1 |=(1 << 3); // output compare preload enable
    TIM3->CCMR1 &= ~(1U << 16); //0
    TIM3->CCMR1 &= ~(0xFU << 4);
    TIM3->CCMR1 |= (0x6U << 4); // mode 1 enable
    TIM3->CCER |= (1U << 0);
//  TIM3->CCR1  = 16000; //duty cycle

    //PWM FOR PB0 TIM3_CH3
    TIM3->CCMR2 |=(1 << 3); // output compare preload enable
    TIM3->CCMR2 &= ~(1U << 16); //0
    TIM3->CCMR2 &= ~(0xFU << 4);
    TIM3->CCMR2 |= (0x6U << 4); // mode 1 enable
```

bsp.c

```
63      TIM3->CCER |= (1U << 8);
64  //  TIM3->CCR3  = 16000; //duty cycle
65
66
67      TIM3->CR1 |= (1 << 0);//    tım3 enable
68
69      NVIC_SetPriority(TIM3_IRQn,3);
70      NVIC_EnableIRQ(TIM3_IRQn);
71
72  }
73
74
75  void TIM3_IRQHandler(void){
76
77      ADC1->CR |= (1U << 2);//Bit 2 ADSTART: ADC start conversion command
78
79      while(0 == (ADC1->ISR & (1U << 2)));
80
81      data = ADC1->DR;
82
83      TIM3->CCR1 = data*62;
84      TIM3->CCR3 = 16000 - (data*62);
85
86      TIM3->SR &= ~(1U << 0); //clear update status register
87
88  }
89
```

main.c

```
 1  /*
 2   * main.c
 3   *
 4   * author: Mehmet Akif Gümüş
 5   */
 6
 7  #include "bsp.h"
 8
 9
10  int main(void) {
11
12      BSP_system_init();
13
14      while(1){
15  }
16      return 0;
17  }
18
19
20
21
22
23
24
```

-APPENDIX B-

ADC.h

```
/*
 * ADC.h
 *
 *  Created on: 5 Oca 2021
 *      author:MEHMET AKİF GÜMÜŞ-171024027
 */

#ifndef ADC_H_
#define ADC_H_

#include "stm32g0xx.h"


void init_ADC();

#endif /* ADC_H_ */

```

ADC.c

```
/*
 * ADC.c
 *
 *  Created on: 5 Oca 2021
 *      author:MEHMET AKİF GÜMÜŞ-171024027
 */

#include "ADC.h"


void init_ADC(void){

    RCC->IOPENR |= (1U << 0);
    RCC->APBENR2 |= (1U << 20);// enable ADC clock

    //setup PA5 as analog
    GPIOA->MODER &= ~(3U << 2*5);
    GPIOA->MODER |= (3U << 2*5);

    ADC1->CR |= (1U << 28); //ADC voltage regulator enabled

    for(uint32_t i=0; i> 0xFFFF; i++);

    ADC1->CR |= (1U << 31);//ADC calibration enabled

    while(0 != (ADC1->CR & (1U << 31)));//wait until completion


    ADC1->IER |= (1U << 2);//End of conversion interrupt enable
    ADC1->CFGR1 |= (2U << 3);// 10: 8 bits

    /*single conversion mode*/

    ADC1->SMPR |= (5U << 0);//Sampling time selection 1 as 101:  39.5  ADC clock cycles
    ADC1->CHSELR |= (1U << 5);//1: Input Channel-5 is selected for conversion
    ADC1->CR |= (1U << 0); //ADC enable command

    while(0 == (ADC1->ISR & (1U << 0)));// 1: ADC is ready to start conversion


}

```

bsp.h

```h
#ifndef BSP_H_
#define BSP_H_

#include "stm32g0xx.h"
#include "ADC.h"
#include "display.h"


void BSP_system_init();


void init_timer1();
void init_timer3();
void BSP_button_init();


#endif

```

bsp.c

```c
/*
 * bps.c
 *
 *  Created on: 5 Oca 2021
 *      author:MEHMET AKİF GÜMÜŞ-171024027
 */


#include "bsp.h"

uint32_t data;
volatile int counter = 0;

void BSP_system_init(void){

    __disable_irq();

        init_timer1();
        init_timer3();
        init_ADC();
        init_SSD();
//      displayID_SSD();
        BSP_button_init();

    __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

    /*10 ms interrupt    */

    TIM1->PSC=10;
    TIM1->ARR=1600;

    TIM1->DIER |= (1 << 0);// update interrupt enable
    TIM1->CR1 |= (1 << 0);//    tim1 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)
{
    ADC1->CR |= (1U << 2);//Bit 2 ADSTART: ADC start conversion command

    while(0 == (ADC1->ISR & (1U << 2)));

    data = ADC1->DR;

    if(data > 243){
        for(uint32_t i=0; i> 0xFFFFFF; i++);
        counter++;
        utility_SSD((float)counter);
```

```
63      }
64
65
66      TIM1->SR &= ~(1U<<0); //clear update status register
67
68  }
69
70  void init_timer3(void){
71
72      RCC->APBENR1 |= (1U<< 1);// enable time3 module clock
73
74      TIM3->CR1=0;// zero out the control register just in case
75      TIM3->CR1 |= (1U << 7); // ARPE
76      TIM3->CNT=0;// zero out counter
77
78      /*10 Msecond interrupt   */
79
80      TIM3->PSC = 10;
81      TIM3->ARR = 160;
82
83      TIM3->DIER |= (1 << 0);// update interrupt enable
84      TIM3->CR1 |= (1 << 0);//    tım3 enable
85
86      NVIC_SetPriority(TIM3_IRQn,3);
87      NVIC_EnableIRQ(TIM3_IRQn);
88
89  }
90
91  void TIM3_IRQHandler(void){
92
93      display_SSD();
94
95      TIM3->SR &= ~(1U << 0); //clear update status register
96  }
97
98
99
100 void BSP_button_init(void){
101         /* Enable GPIOA clock */
102         RCC->IOPENR |= (1U << 0);
103
104         /* Setup PA1 as input */
105         GPIOA->MODER &= ~(3U << 2*1);
106         GPIOA->PUPDR |= (2U << 2*1); // Pull-down mode
107
108         /*setup interrrupts for inputs*/
109         EXTI->EXTICR[0] |= (0U << 8*1);//PA1     EXTI1 mux ta PA1 için EXTICR0'ın 9.biti 0
    yapıldı
110
111         /* MASK*/
112         EXTI->IMR1 |= (1U << 1);
113
114         /*rising edge*/
115         EXTI->RTSR1 |= (1U << 1);
116
117         /*NVIC*/
118         NVIC_SetPriority(EXTI0_1_IRQn,0);   // buton interruptı PA1 için EXTI1 in
119                                             //içerisinde olduğundan EXTI0_1_IRQn
    kullanıldı.
120         NVIC_EnableIRQ(EXTI0_1_IRQn);   //nvic enabled
121 }
122
```



```
123 void EXTI0_1_IRQHandler(void){
124     if (EXTI->RPR1 & (1U << 1)){
125     counter=0;
126     utility_SSD((float)counter);
127     }
128     EXTI->RPR1 |= (1U << 1);
129
130     for(uint32_t i=0; i> 0xFFFFFFFF; i++);
131
132
133 }
134
135
```

display.h

```c
/*
 * display.h
 *
 *  Created on: Dec 19, 2020
 *      author:MEHMET AKİF GÜMÜŞ-171024027
 */

#ifndef DISPLAY_H_
#define DISPLAY_H_

#include "bsp.h"

typedef struct{
    uint8_t Digits[4];

}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();


/*
 * 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);


#endif /* DISPLAY_H_ */

```

```c
/*
 * display.c
 *
 *author:MEHMET AKİF GÜMÜŞ-171024027
 */

#include "display.h"

extern void main();


void init_SSD(){

    RCC->IOPENR |= (3U << 0);

    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

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

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

    GPIOB->MODER &= ~(3U << 2*0);
    GPIOB->MODER |= (1U << 2*0);//PB0 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(i == 1){
            GPIOA->ODR |= (1U << 7); //PA7
            GPIOB->ODR &= ~(1U << 6); //PB6
```

```
 63             GPIOB->ODR &= ~(1U << 7); //PB7
 64             GPIOB->ODR &= ~(1U << 1); //PB1
 65             printDigit_SSD(Display.Digits[0]);
 66 
 67 
 68         }
 69         else if(i == 10){
 70             GPIOA->ODR &= ~(1U << 7); //PA7
 71             GPIOB->ODR |= (1U << 6); //PB6
 72             GPIOB->ODR &= ~(1U << 7); //PB7
 73             GPIOB->ODR &= ~(1U << 1); //PB1
 74             printDigit_SSD(Display.Digits[1]);
 75 
 76 
 77         }
 78 
 79         else if(i == 20){
 80             GPIOA->ODR &= ~(1U << 7); //PA7
 81             GPIOB->ODR &= ~(1U << 6); //PB6
 82             GPIOB->ODR |= (1U << 7); //PB7
 83             GPIOB->ODR &= ~(1U << 1); //PB1
 84             printDigit_SSD(Display.Digits[2]);
 85 
 86         }
 87         else if(i == 30){
 88             GPIOA->ODR &= ~(1U << 7); //PA7
 89             GPIOB->ODR &= ~(1U << 6); //PB6
 90             GPIOB->ODR &= ~(1U << 7); //PB7
 91             GPIOB->ODR |= (1U << 1); //PB1
 92             printDigit_SSD(Display.Digits[3]);
 93 
 94         }
 95         else if(i == 40) i = 0;
 96 
 97         i++;
 98 
 99 
100 }
101 
102 void printDigit_SSD(uint8_t x){
103 
104         switch(x){
105         case 0:     //0
106 
107             GPIOB->ODR &= ~( 1U << 3); // PB3
108             GPIOA->ODR &= ~( 1U << 0); // PA0
109             GPIOB->ODR &= ~( 1U << 4); // PB4
110             GPIOA->ODR &= ~( 1U << 4); // PA4
111             GPIOB->ODR &= ~( 1U << 0); // PB0
112             GPIOA->ODR &= ~( 1U << 12); // PA12
113             GPIOA->ODR |=  ( 1U << 11); // PA11
114 
115             break;
116 
117         case 1:     //1
118             GPIOB->ODR |= ( 1U << 3); // PB3
119             GPIOA->ODR &= ~( 1U << 0); // PA0
120             GPIOB->ODR &= ~( 1U << 4); // PB4
121             GPIOA->ODR |= ( 1U << 4); // PA4
122             GPIOB->ODR |= ( 1U << 0); // PB0
123             GPIOA->ODR |= ( 1U << 12); // PA12
124             GPIOA->ODR |= ( 1U << 11); // PA11
```

```
125
126             break;
127
128         case 2:     //2
129             GPIOB->ODR &= ~( 1U << 3); // PB3
130             GPIOA->ODR &= ~( 1U << 0); // PA0
131             GPIOB->ODR |=  ( 1U << 4); // PB4
132             GPIOA->ODR &= ~( 1U << 4); // PA4
133             GPIOB->ODR &= ~( 1U << 0); // PB0
134             GPIOA->ODR |=  ( 1U << 12); // PA12
135             GPIOA->ODR &= ~( 1U << 11); // PA11
136
137             break;
138
139         case 3:     //3
140
141             GPIOB->ODR &= ~( 1U << 3); // PB3
142             GPIOA->ODR &= ~( 1U << 0); // PA0
143             GPIOB->ODR &= ~( 1U << 4); // PB4
144             GPIOA->ODR &= ~( 1U << 4); // PA4
145             GPIOB->ODR |= ( 1U << 0); // PB0
146             GPIOA->ODR |=  ( 1U << 12); // PA12
147             GPIOA->ODR &= ~( 1U << 11); // PA11
148
149             break;
150
151         case 4:     //4
152             GPIOB->ODR |=  ( 1U << 3); // PB3
153             GPIOA->ODR &= ~( 1U << 0); // PA0
154             GPIOB->ODR &= ~( 1U << 4); // PB4
155             GPIOA->ODR |=  ( 1U << 4); // PA4
156             GPIOB->ODR |= ( 1U << 0); // PB0
157             GPIOA->ODR &= ~( 1U << 12); // PA12
158             GPIOA->ODR &= ~( 1U << 11); // PA11
159
160             break;
161
162         case 5:     //5
163
164             GPIOB->ODR &= ~( 1U << 3); // PB3
165             GPIOA->ODR |=  ( 1U << 0); // PA0
166             GPIOB->ODR &= ~( 1U << 4); // PB4
167             GPIOA->ODR &= ~( 1U << 4); // PA4
168             GPIOB->ODR |= ( 1U << 0); // PB0
169             GPIOA->ODR &= ~( 1U << 12); // PA12
170             GPIOA->ODR &= ~( 1U << 11); // PA11
171
172             break;
173
174         case 6:     //6
175             GPIOB->ODR &= ~( 1U << 3); // PB3
176             GPIOA->ODR |=  ( 1U << 0); // PA0
177             GPIOB->ODR &= ~( 1U << 4); // PB4
178             GPIOA->ODR &= ~( 1U << 4); // PA4
179             GPIOB->ODR &= ~( 1U << 0); // PB0
180             GPIOA->ODR &= ~( 1U << 12); // PA12
181             GPIOA->ODR &= ~( 1U << 11); // PA11
182
183             break;
184
185         case 7:     //7
186
```

```
187             GPIOB->ODR &= ~( 1U << 3); // PB3
188             GPIOA->ODR &= ~( 1U << 0); // PA0
189             GPIOB->ODR &= ~( 1U << 4); // PB4
190             GPIOA->ODR |=  ( 1U << 4); // PA4
191             GPIOB->ODR |= ( 1U << 0); // PB0
192             GPIOA->ODR |=  ( 1U << 12); // PA12
193             GPIOA->ODR |=  ( 1U << 11); // PA11
194 
195             break;
196 
197         case 8:     //8
198 
199             GPIOB->ODR &= ~( 1U << 3); // PB3
200             GPIOA->ODR &= ~( 1U << 0); // PA0
201             GPIOB->ODR &= ~( 1U << 4); // PB4
202             GPIOA->ODR &= ~( 1U << 4); // PA4
203             GPIOB->ODR &= ~( 1U << 0); // PB0
204             GPIOA->ODR &= ~( 1U << 12); // PA12
205             GPIOA->ODR &= ~( 1U << 11); // PA11
206 
207             break;
208 
209         case 9:     //9
210             GPIOB->ODR &= ~( 1U << 3); // PB3
211             GPIOA->ODR &= ~( 1U << 0); // PA0
212             GPIOB->ODR &= ~( 1U << 4); // PB4
213             GPIOA->ODR &= ~( 1U << 4); // PA4
214             GPIOB->ODR |= ( 1U << 0); // PB0
215             GPIOA->ODR &= ~( 1U << 12); // PA12
216             GPIOA->ODR &= ~( 1U << 11); // PA11
217 
218             break;
219         }
220 }
221 
222 
223 //void displayID_SSD(void){
224 //  Display.Digits[0]= 7;
225 //  Display.Digits[1]= 2;
226 //  Display.Digits[2]= 7;
227 //  Display.Digits[3]= 1;
228 //
229 //}
230 
231 
232 
233 void utility_SSD(float var){
234 
235     int number = (int)var;
236 
237 
238     int temp = number / 10;
239     Display.Digits[0] = (uint8_t)(number - (temp*10));
240 
241     temp = number / 100;
242     Display.Digits[1] = (uint8_t)((number - (temp * 100)) / 10);
243 
244     temp = number / 1000;
245     Display.Digits[2] = (uint8_t)((number - (temp * 1000)) / 100);
246 
247     temp = number / 10000;
248     Display.Digits[3] = (uint8_t)((number - (temp * 10000)) / 1000);
```

main.c

```c
/*
 * main.c
 *
 * author:MEHMET AKİF GÜMÜŞ-171024027
 */

#include "bsp.h"

int main(void) {

    BSP_system_init();

    while(1){
}
    return 0;
}
```

4. REFERENCES

- https://skybluetrades.net/stm32-timer-adc-dma-1/#6-set-adc-sample-time
- https://deepbluembedded.com/stm32-adc-tutorial-complete-guide-with-examples/
- https://www.st.com/content/ccc/resource/training/technical/product_training/group0/c3/a7/16/ca/55/fb/4a/47/STM32G0-Analog-ADC/files/STM32G0-Analog-ADC-ADC.pdf/jcr:content/translations/en.STM32G0-Analog-ADC-ADC.pdf
- https://www.youtube.com/watch?v=3WyJc8yPCT0&feature=youtu.be&ab_channel=FurkanCayci
- https://embedds.com/introducing-to-stm32-adc-programming-part2/
- https://en.wikipedia.org/wiki/Potentiometer#:~:text=A%20potentiometer%20is%20a%20three,forms%20an%20adjustable%20voltage%20divider.&text=Potentiometers%20are%20commonly%20used%20to,volume%20controls%20on%20audio%20equipment.
- https://techterms.com/definition/microphone