GEBZE TECNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC 334 - Project #3

A digital voice recorder

Project #3 REPORT

Preparer:
1) 171024027 – Mehmet Akif GÜMÜŞ

1. INRODUCTION:

Main objective of this project/final is to create a digital voice recorder that can record your voice, playback a selected voice recording, and delete single or all recording data. During this project, you will use various modules such as Timer, PWM, ADC, and External Interrupts.

2. Technical requirements:

- Written in C. No HAL or equivalent libraries.
- Connect a microphone to record your voice. Keep in mind that if this microphone does not have an on-board amplifier, you will need to build one yourself.
- Build an amplifier and connect a speaker with variable pot to playback the recordings.
- Connect 2 x 24LC512 EEPROMs on the same I2C bus. Keep in mindm when wiring the bus will require pull-up resistors on both lines, and each of these devices need dierent address to communicate.
- You should be able to at least record 4 tracks with 5 seconds each. 5 seconds should be fixed, but if you can fit more tracks that is fine
    - Calculate the maximum datasize for two EEPROMs for keeping your data and create a table of how many seconds can be recorded with dierent data rates. Pick one that will fit the requirement.
- A keypad should be attached to operate the device.
    - Assign a key for recording a voice. The recording will go for 5 seconds and automatically stop/save it. After the track is played, it will stop and go back to IDLE state.
        - Pressing any other button should not have any effect
    - Assign first 4+ number keys for track select when not recording. For example pressing 1 will select the first track, pressing 2 will select the second track, etc. This key press will not do anything else.
    - Assign a key for playing/pausing the selected track when not recording. After the track is played, it will stop and go back to IDLE state.
        - Pressing any other button should not have any effect.
    - Assign a key for deleting the selected track. After the track is deleted, it will go back to IDLE state.
    - Assign a key for seeing the track status. After the key is pressed, 7SD shows the number of available tracks.
- A 7SD should be attached to display the operations and status.
- You should have multiple states, some of which include:
    - START state which only happens when the board powers up 7SD should show your ID (first 2 and last 2 digits)
    - IDLE state which displays IdLE on the 7SD and does not do anything else. (waiting for track select or record start)
    - FULL state which displays FuLL on the 7SD and prevents going into RECORD state.

- RECORD state where the 7SD shows rcd and a count down from 5 seconds indicating the recording. (i.e. rcd3, rcd2)
- PLAYBACK state where the 7SD shows PLb and the track being played back (i.e. PLb2, PLb1)
- STATUS state where the 7SD shows Ava the number of available tracks. (i.e. Ava3, Ava0)

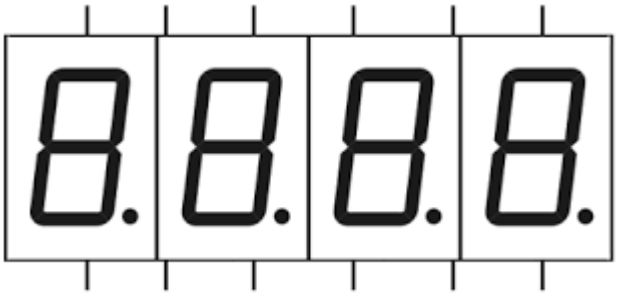
- If no button is pressed for 10 seconds, the device should go back to IDLE state.

THEORETICAL RESEARCH

• Seven-segment display

A **seven-segment display** is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.



Şekil 1. example of 4-digit 7 segment display

• KeyPAD

The 4*4 matrix keypad usually is used as input in a project. It has 16 keys in total, which means the same input values.

The SunFouner 4*4 Matrix Keypad Module is a matrix non- encoded keypad consisting of 16 keys in parallel. The keys of each row and column are connected through the pins outside – pin Y1-Y4 as labeled beside control the rows, when X1-X4, the columns.

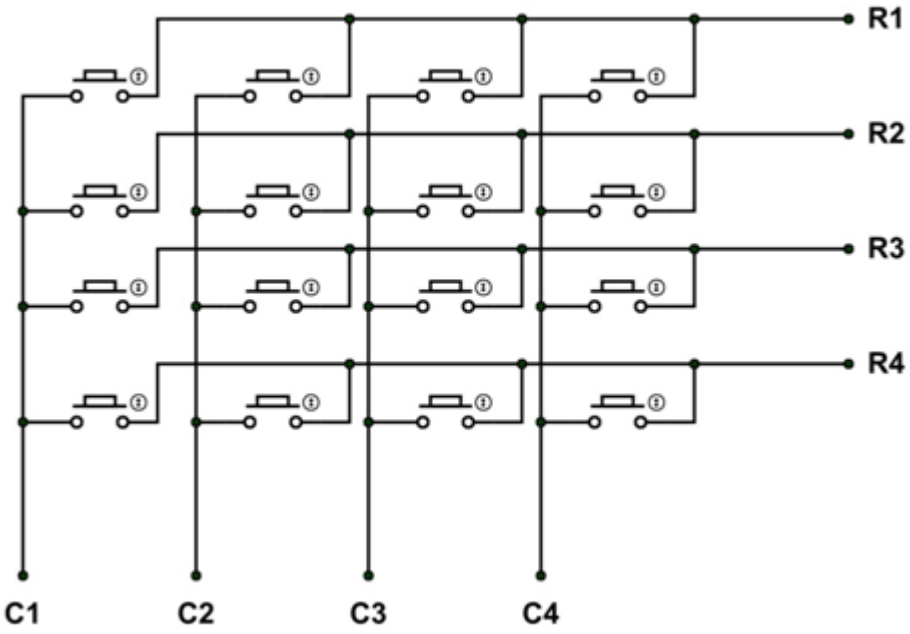
How it works

First test whether any key is pressed down. Connect power to rows, so they are High level. Then set all the rows Y1-Y4 as Low and then detect the status of the columns. Any column of Low indicates there is key pressing and that the key is among the 4 keys of the column. If all columns are High, it means no key is pressed down.

Next, locate the key. Since the column in which the pressed key lies is identified, knowing the line would finalize the testing. Thus, set the rows as Low in turns until any is unveiled accordingly – other rows will still be High. Now the row can be identified. Detect the status of each column in turns. The

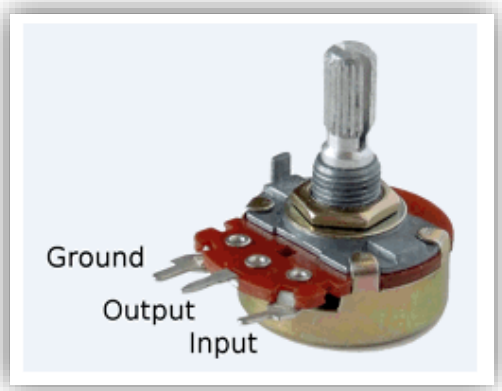
column tested Low is the one intersecting with the line – their cross point is just the key pressed.

The schmatic diagram:



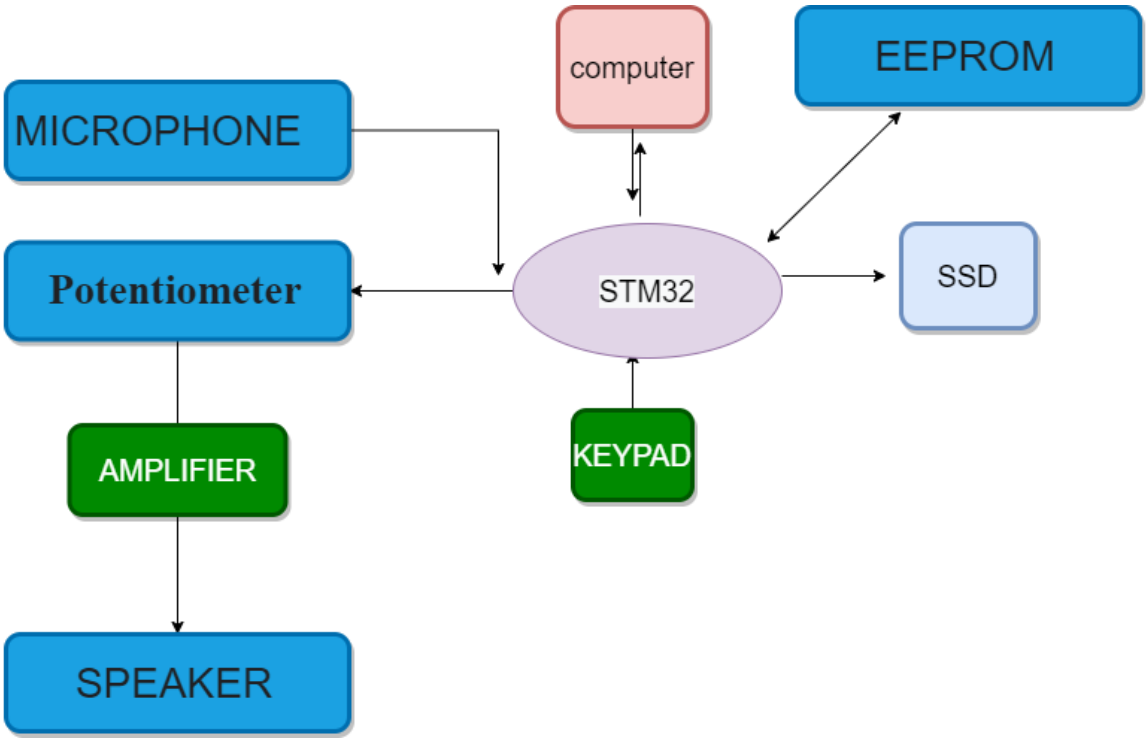
- ## Potentiometer:

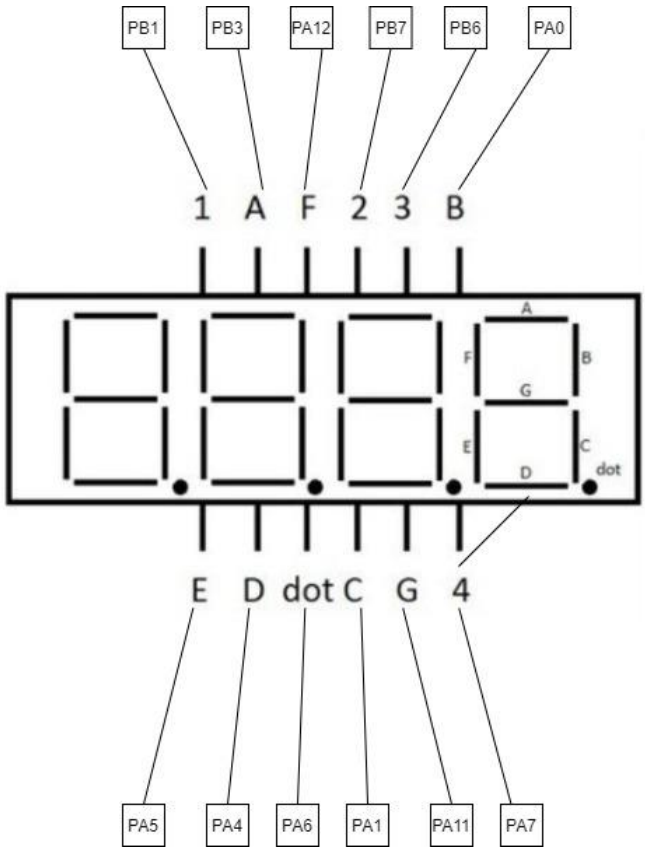
A **potentiometer** is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a **variable resistor** or **rheostat**.

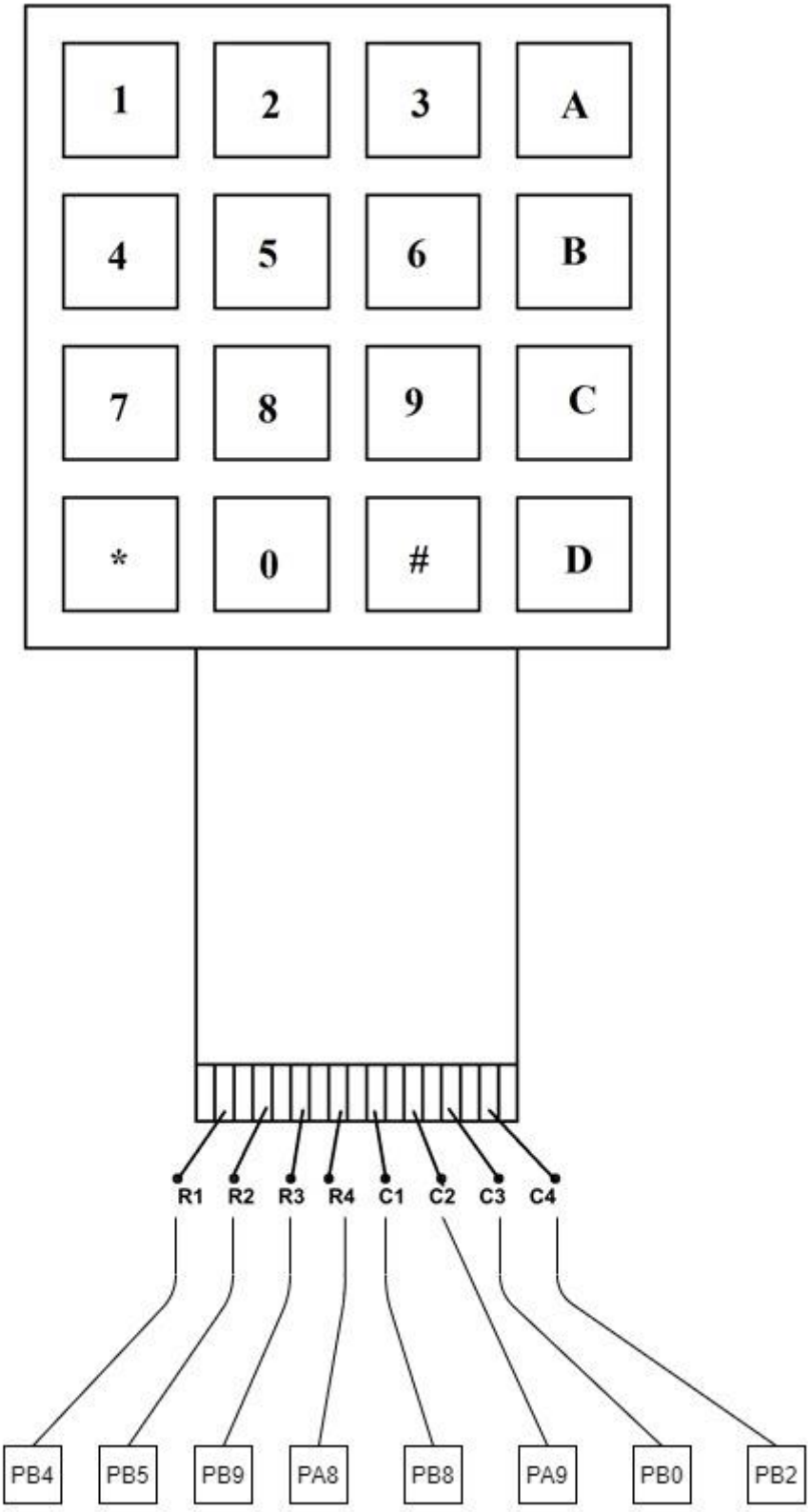


The measuring instrument called a potentiometer is essentially a voltage divider used for measuring electric potential (voltage); the component is an implementation of the same principle, hence its name.

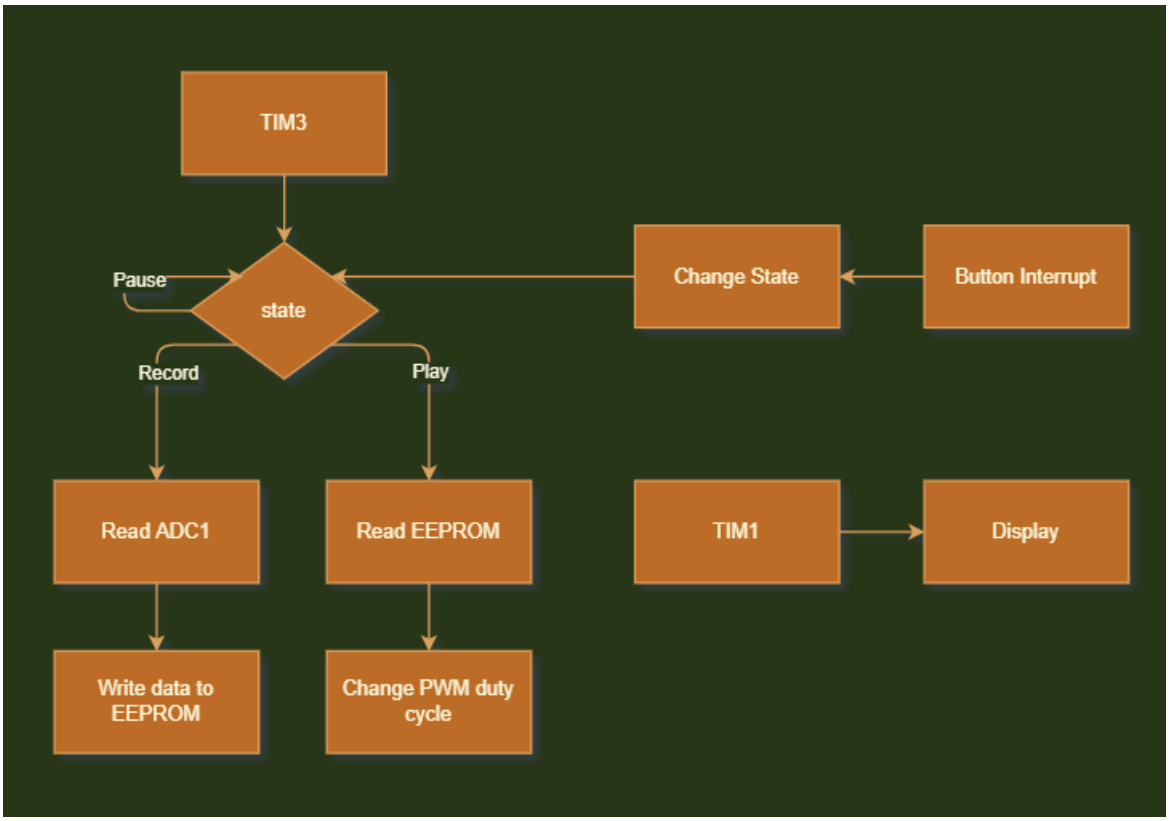
3. BLOCK DIAGRAM

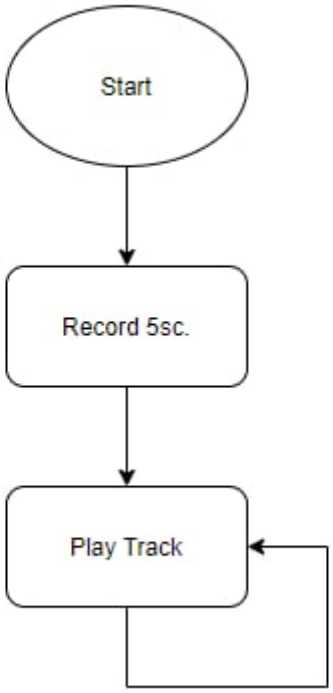





1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D
R1
R2
R3
R4
C1
C2
C3
C4
PB4
PB5
PB9
PA8
PB8
PA9
PB0
PB2

4. FLOWCHART

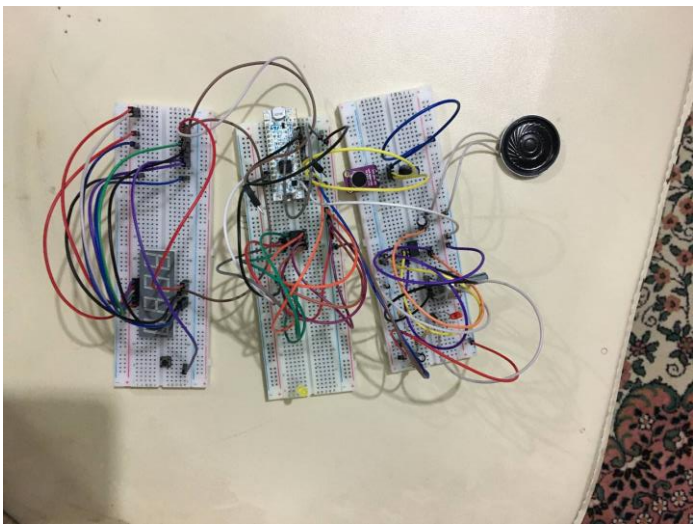

TIM3
Pause
state
Change State
Button Interrupt
Record
Play
Read ADC1
Read EEPROM
TIM1
Display
Write data to EEPROM
Change PWM duty cycle

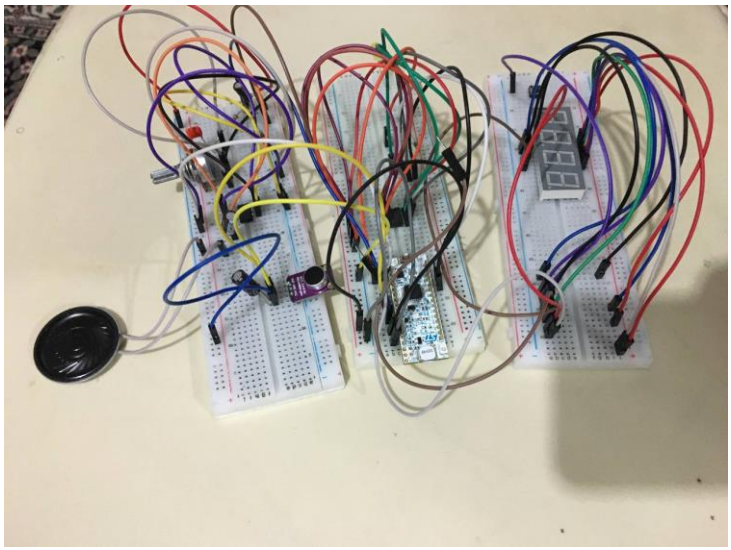

Start
Record 5sc.
Play Track

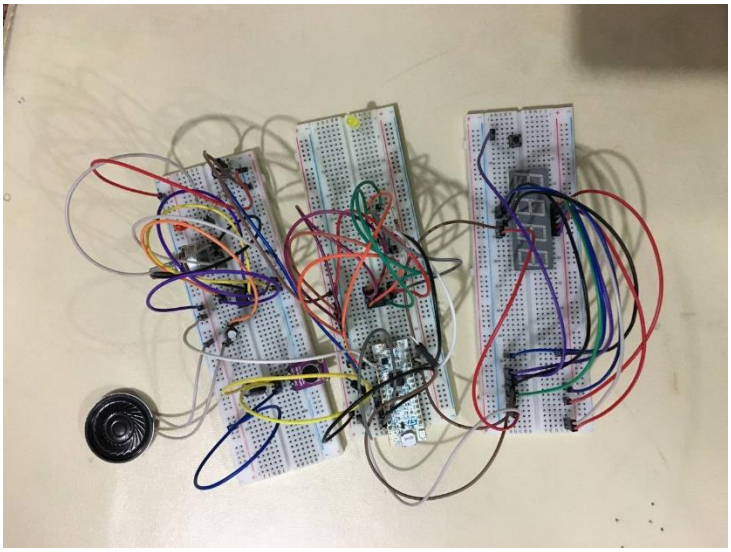
5. TASK

- nucleo-32 card schematıcs examıned
- The seven segment display block diagram was examined
- Created flowchart
- Created blockdiagram
- Necessary connections are made on the breadboard
- Keypad and Seven Segment Display connection has been made
- Determine interrupts
- Utilize interrupts for all functionality
- School ID shows by `displayID_SSD` func
- When keys are entered ssd shows the numbers
- No more than 4 digits by `Keypad_data` func
- ABCDEF keys use as by “calc.c”
- Floating point number system by `utility_SSD` func
- Negative numbers have negative sign by `utility_SSD` func
- If the result overflows display shows OuFL by `overflow_SSD` func
- If the operation is invalid display shows Invd by `invalid_SSD` func

6. PROJECT SETUP WITH PICTURES:







7. PARTS LIST:

MATERIAL	PRICE
LM386 AMPLIFIER	1,25
MIC MAX4466	20
ROHS 8 OHM SPEAKER	8
7SD	7,5
STM32 G031K8	95
OTHER COMP.	45

8. CONCLUSION:

The purpose of this experiment is to create a digital voice recorder that can record your voice, playback a selected voice recording, and delete single or all recording data.
In this problem, we converted the analog values we got from the microphone into digital data with ADC (installation steps are given below). Then we wrote these data to eeprom. Then we read the data on the eeprom. The read data were used to set the Pwm signal that we installed in TIM3. In order to adjust the dutycycle of the PWM signal, the available values were written to the CRRx value. Dutycycles between 0-100% were obtained thanks to these values. The obtained PWM signal was given to the circuit we set up from the PA6 pin. This signal reaches the speaker by passing through the circuit elements in the circuit.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

In this problem, we had a hard time establishing the circuit. Since the pins of some elements are not soldered, we had to solder. Many short circuits and non-contact were encountered in the circuit. There was a lot of noise as there was no filtering on the microphone. While the microphone was idle it gave values around 128/255. To overcome this, values higher than 240 were considered. Voice was recorded for 5 seconds. The received voice recording was played for 5 seconds. While recording sound, the sound given from the speaker could not be understood because the sample time, Timer, processor frequency, Adc clock calculations were not done correctly. However, the peak points of the sound were heard correctly from the speaker. Elements such as keypad and 7SD specified in the project could not be used because other operations took a lot of time and time was insufficient. However, the use of keypad and 7SD elements has been successfully used in previous projects.

9. REFERANCES:

- https://www.instructables.com/Using-a-4-digit-7-segment-display-with-arduino/
- https://en.wikipedia.org/wiki/Seven-segment_display
- https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet
- http://wiki.sunfounder.cc/index.php?title=4X4_Matrix_Keypad_Module
- https://www.st.com/resource/en/application_note/cd00211314-how-to-get-the-best-adc-accuracy-in-stm32-microcontrollers-stmicroelectronics.pdf
- https://cdn-shop.adafruit.com/datasheets/MAX4465-MAX4469.pdf
- https://www.maximintegrated.com/en/products/analog/audio/MAX4466.html
- https://market.samm.com/raspberry-pi-electret-mikrofon-amfisi-max4466

Appendix Codes:

```c
/*
 * ADC.c
 *
 *  Created on: Jan 8, 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 << 11);//1: End of calibration interrupt enabled
    ADC1->IER |= (1U << 2);//End of conversion interrupt enable

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

    ADC1->SMPR |= (7U << 0);//Sampling time selection 111 as 1: 160.5 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


}

```

```c
/*
 * ADC.h
 *
 *  Created on: Jan 8, 2021
 *      Author:
 */

#ifndef ADC_H_
#define ADC_H_

#include "main.h"

void init_ADC();

#endif /* ADC_H_ */

```

```c
/*
 * BSP.h
 *
 *  Created on: Jan 8, 2021
 *      Author:
 */

#ifndef BSP_H_
#define BSP_H_


#include "ADC.h"
#include "display.h"
#include "eeprom.h"


void BSP_init();

void TIM1_init();
void TIM3_init();
void soundfunc(uint8_t a, uint8_t *ARR);

uint8_t sound_buffer[100];

#endif /* BSP_H_ */

```

```c
/*
 * BSP.c
 *
 *  Created on: Jan 8, 2021
 *      Author:
 */

#include "BSP.h"
#include "string.h"
#include "stdlib.h"

uint32_t k=0;
int sayac;

void BSP_init(){

    __disable_irq();

//      SSD_init();
        init_ADC();
        TIM1_init();
        TIM3_init();
        I2C_init_();
        sayac = 0;
    __enable_irq();


}


void TIM1_init(){

    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 sayac

    /*0.1 ms interrupt   */

    TIM1->PSC=10;
    TIM1->ARR=160;

    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)
{

    display_SSD();

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

}

void TIM3_init(void){

```

```
63      RCC->IOPENR |= 7; /*????????????????*/
64
65      //setup PA6 as AF2
66      GPIOA->MODER &= ~(3U << 2*6);
67      GPIOA->MODER |= (2U << 2*6);
68
69      // choose AF2 from mux
70      GPIOA->AFR[0] &= ~(0xFU << 4*6);
71      GPIOA->AFR[0] |= (1U << 4*6);
72
73      ///////////////
74      RCC->APBENR1 |= (1U<< 1);// enable timer3 clock
75
76      TIM3->CR1=0;// zero out the control register just in case
77      TIM3->CR1 |= (1U << 7); // ARPE
78      TIM3->CNT=0;// zero out sayac
79
80      //100us -> 0.10ms
81      TIM3->PSC = 1;
82      TIM3->ARR = 255;
83      TIM3->DIER |= (1 << 0);// update interrupt enable
84
85      ////////////
86
87      //PWM FOR PA6 TIM3_CH1
88      TIM3->CCMR1 |=(1 << 3); // output compare preload enable
89      TIM3->CCMR1 &= ~(1U << 16); //0
90      TIM3->CCMR1 &= ~(0xFU << 4);
91      TIM3->CCMR1 |= (0x6U << 4); // mode 1 enable
92      TIM3->CCER |= (1U << 0);
93
94      TIM3->CCR1 = 8;
95
96      TIM3->CR1 |= (1 << 0);//    tim3 enable
97
98      NVIC_SetPriority(TIM3_IRQn, 2);
99      NVIC_EnableIRQ(TIM3_IRQn);
100
101 }
102
103 void TIM3_IRQHandler(void){
104
105     TIM3->SR &= ~(1U << 0); //clear update status register
106
107     ADC1->CR |= (1U << 2);//Bit 2 ADSTART: ADC start conversion command
108
109     while(0 == (ADC1->ISR & (1U << 2)));
110
111
112     uint8_t transporter;
113     if(sayac == 10000){
114         k = 1;
115         sayac = 0;
116     }
117     switch(k){
118     case 0:
119         //ADC
120         if(sayac < 10000){
121             transporter = (uint8_t)ADC1->DR;
122
123                 transporter = (uint8_t)((transporter-127)*2);
124                 soundfunc(transporter, sound_buffer);
```

```
125
126             if(!(sayac % 100)){
127                 write_ee((uint16_t)sayac, sound_buffer, 100);
128                 for(int i = 0xFFFF; i>0; i--);
129             }
130             sayac++;
131         }
132         break;
133     case 1:
134         //PWM
135         if(sayac<10000){
136 
137                 read_ee((uint16_t)sayac, &transporter, 1);
138 
139                 TIM3->CCR1 = (uint32_t)(transporter);
140 
141 
142             sayac++;
143         }
144         break;
145     case 3:
146         //idle();
147         break;
148 
149     }
150 
151 
152 }
153 
154 
155 void soundfunc(uint8_t a, uint8_t *ARR){
156     static int i = 0;
157 
158         if (i<100){
159             ARR[i] = a;
160             i++;
161         }
162         else{
163         i = 0;
164         }
165 
166 
167 
168 }
169 
170 
```

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

#include "display.h"

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(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

```

```
 63         }
 64         else if(i == 10){
 65             GPIOA->ODR &= ~(1U << 7); //PA7
 66             GPIOB->ODR |= (1U << 6); //PB6
 67             GPIOB->ODR &= ~(1U << 7); //PB7
 68             GPIOB->ODR &= ~(1U << 1); //PB1
 69             printDigit_SSD(Display.Digits[1]);
 70             GPIOA->ODR |= ( 1U << 6); // PA6
 71 
 72         }
 73 
 74         else if(i == 20){
 75             GPIOA->ODR &= ~(1U << 7); //PA7
 76             GPIOB->ODR &= ~(1U << 6); //PB6
 77             GPIOB->ODR |= (1U << 7); //PB7
 78             GPIOB->ODR &= ~(1U << 1); //PB1
 79             printDigit_SSD(Display.Digits[2]);
 80             GPIOA->ODR |= ( 1U << 6); // PA6
 81 
 82         }
 83         else if(i == 30){
 84             GPIOA->ODR &= ~(1U << 7); //PA7
 85             GPIOB->ODR &= ~(1U << 6); //PB6
 86             GPIOB->ODR &= ~(1U << 7); //PB7
 87             GPIOB->ODR |= (1U << 1); //PB1
 88             printDigit_SSD(Display.Digits[3]);
 89             GPIOA->ODR |= ( 1U << 6); // PA6
 90 
 91         }
 92         else if(i == 40) i = 0;
 93 
 94         i++;
 95 
 96 
 97 
 98 
 99 }
100 
101 void printDigit_SSD(uint8_t x){
102 
103         switch(x){
104         case 0:     //0
105 
106             GPIOB->ODR &= ~( 1U << 3); // PB3
107             GPIOA->ODR &= ~( 1U << 0); // PA0
108             GPIOA->ODR &= ~( 1U << 1); // PA1
109             GPIOA->ODR &= ~( 1U << 4); // PA4
110             GPIOA->ODR &= ~( 1U << 5); // PA5
111             GPIOA->ODR &= ~( 1U << 12); // PA12
112             GPIOA->ODR |=  ( 1U << 11); // PA11
113 
114             break;
115 
116         case 1:     //1
117             GPIOB->ODR |= ( 1U << 3); // PB3
118             GPIOA->ODR &= ~( 1U << 0); // PA0
119             GPIOA->ODR &= ~( 1U << 1); // PA1
120             GPIOA->ODR |= ( 1U << 4); // PA4
121             GPIOA->ODR |= ( 1U << 5); // PA5
122             GPIOA->ODR |= ( 1U << 12); // PA12
123             GPIOA->ODR |= ( 1U << 11); // PA11
124 
```

```
125             break;
126 
127         case 2:     //2
128             GPIOB->ODR &= ~( 1U << 3); // PB3
129             GPIOA->ODR &= ~( 1U << 0); // PA0
130             GPIOA->ODR |=  ( 1U << 1); // PA1
131             GPIOA->ODR &= ~( 1U << 4); // PA4
132             GPIOA->ODR &= ~( 1U << 5); // PA5
133             GPIOA->ODR |=  ( 1U << 12); // PA12
134             GPIOA->ODR &= ~( 1U << 11); // PA11
135 
136             break;
137 
138         case 3:     //3
139 
140             GPIOB->ODR &= ~( 1U << 3); // PB3
141             GPIOA->ODR &= ~( 1U << 0); // PA0
142             GPIOA->ODR &= ~( 1U << 1); // PA1
143             GPIOA->ODR &= ~( 1U << 4); // PA4
144             GPIOA->ODR |=  ( 1U << 5); // PA5
145             GPIOA->ODR |=  ( 1U << 12); // PA12
146             GPIOA->ODR &= ~( 1U << 11); // PA11
147 
148             break;
149 
150         case 4:     //4
151             GPIOB->ODR |=  ( 1U << 3); // PB3
152             GPIOA->ODR &= ~( 1U << 0); // PA0
153             GPIOA->ODR &= ~( 1U << 1); // PA1
154             GPIOA->ODR |=  ( 1U << 4); // PA4
155             GPIOA->ODR |=  ( 1U << 5); // PA5
156             GPIOA->ODR &= ~( 1U << 12); // PA12
157             GPIOA->ODR &= ~( 1U << 11); // PA11
158 
159             break;
160 
161         case 5:     //5
162 
163             GPIOB->ODR &= ~( 1U << 3); // PB3
164             GPIOA->ODR |=  ( 1U << 0); // PA0
165             GPIOA->ODR &= ~( 1U << 1); // PA1
166             GPIOA->ODR &= ~( 1U << 4); // PA4
167             GPIOA->ODR |=  ( 1U << 5); // PA5
168             GPIOA->ODR &= ~( 1U << 12); // PA12
169             GPIOA->ODR &= ~( 1U << 11); // PA11
170 
171             break;
172 
173         case 6:     //6
174             GPIOB->ODR &= ~( 1U << 3); // PB3
175             GPIOA->ODR |=  ( 1U << 0); // PA0
176             GPIOA->ODR &= ~( 1U << 1); // PA1
177             GPIOA->ODR &= ~( 1U << 4); // PA4
178             GPIOA->ODR &= ~( 1U << 5); // PA5
179             GPIOA->ODR &= ~( 1U << 12); // PA12
180             GPIOA->ODR &= ~( 1U << 11); // PA11
181 
182             break;
183 
184         case 7:     //7
185 
186             GPIOB->ODR &= ~( 1U << 3); // PB3
```

```
187             GPIOA->ODR &= ~( 1U << 0); // PA0
188             GPIOA->ODR &= ~( 1U << 1); // PA1
189             GPIOA->ODR |=  ( 1U << 4); // PA4
190             GPIOA->ODR |=  ( 1U << 5); // PA5
191             GPIOA->ODR |=  ( 1U << 12); // PA12
192             GPIOA->ODR |=  ( 1U << 11); // PA11
193 
194             break;
195 
196         case 8:     //8
197 
198             GPIOB->ODR &= ~( 1U << 3); // PB3
199             GPIOA->ODR &= ~( 1U << 0); // PA0
200             GPIOA->ODR &= ~( 1U << 1); // PA1
201             GPIOA->ODR &= ~( 1U << 4); // PA4
202             GPIOA->ODR &= ~( 1U << 5); // PA5
203             GPIOA->ODR &= ~( 1U << 12); // PA12
204             GPIOA->ODR &= ~( 1U << 11); // PA11
205 
206             break;
207 
208         case 9:     //9
209             GPIOB->ODR &= ~( 1U << 3); // PB3
210             GPIOA->ODR &= ~( 1U << 0); // PA0
211             GPIOA->ODR &= ~( 1U << 1); // PA1
212             GPIOA->ODR &= ~( 1U << 4); // PA4
213             GPIOA->ODR |=  ( 1U << 5); // PA5
214             GPIOA->ODR &= ~( 1U << 12); // PA12
215             GPIOA->ODR &= ~( 1U << 11); // PA11
216 
217             break;
218 
219         case 10://A
220 
221             GPIOB->ODR &= ~( 1U << 3); // PB3
222             GPIOA->ODR &= ~( 1U << 0); // PA0
223             GPIOA->ODR &= ~( 1U << 1); // PA1
224             GPIOA->ODR &= ~( 1U << 4); // PA4
225             GPIOA->ODR &= ~( 1U << 5); // PA5
226             GPIOA->ODR |=  ( 1U << 12); // PA12
227             GPIOA->ODR &= ~( 1U << 11); // PA11
228 
229 
230 
231             break;
232 
233         case 11://B
234 
235             GPIOB->ODR |=  ( 1U << 3); // PB3
236             GPIOA->ODR |=  ( 1U << 0); // PA0
237             GPIOA->ODR &= ~( 1U << 1); // PA1
238             GPIOA->ODR &= ~( 1U << 4); // PA4
239             GPIOA->ODR &= ~( 1U << 5); // PA5
240             GPIOA->ODR &= ~( 1U << 12); // PA12
241             GPIOA->ODR &= ~( 1U << 11); // PA11
242 
243 
244             break;
245 
246         case 12://C
247             GPIOB->ODR &= ~( 1U << 3); // PB3
248             GPIOA->ODR |=  ( 1U << 0); // PA0
```

```
249             GPIOA->ODR |=  ( 1U << 1); // PA1
250             GPIOA->ODR &= ~( 1U << 4); // PA4
251             GPIOA->ODR &= ~( 1U << 5); // PA5
252             GPIOA->ODR &= ~( 1U << 12); // PA12
253             GPIOA->ODR |=  ( 1U << 11); // PA11
254 
255             break;
256 
257         case 13://D
258             GPIOB->ODR |=  ( 1U << 3); // PB3
259             GPIOA->ODR &= ~( 1U << 0); // PA0
260             GPIOA->ODR &= ~( 1U << 1); // PA1
261             GPIOA->ODR &= ~( 1U << 4); // PA4
262             GPIOA->ODR &= ~( 1U << 5); // PA5
263             GPIOA->ODR |=  ( 1U << 12); // PA12
264             GPIOA->ODR &= ~( 1U << 11); // PA11
265 
266             break;
267 
268         case 14://E
269             GPIOB->ODR &= ~( 1U << 3); // PB3
270             GPIOA->ODR |=  ( 1U << 0); // PA0
271             GPIOA->ODR |=  ( 1U << 1); // PA1
272             GPIOA->ODR &= ~( 1U << 4); // PA4
273             GPIOA->ODR &= ~( 1U << 5); // PA5
274             GPIOA->ODR &= ~( 1U << 12); // PA12
275             GPIOA->ODR &= ~( 1U << 11); // PA11
276 
277 
278             break;
279 
280         case 15: //F
281             GPIOB->ODR &= ~( 1U << 3); // PB3
282             GPIOA->ODR |=  ( 1U << 0); // PA0
283             GPIOA->ODR |=  ( 1U << 1); // PA1
284             GPIOA->ODR |=  ( 1U << 4); // PA4
285             GPIOA->ODR &= ~( 1U << 5); // PA5
286             GPIOA->ODR &= ~( 1U << 12); // PA12
287             GPIOA->ODR &= ~( 1U << 11); // PA11
288             break;
289 
290 
291         case 30: //u
292             GPIOB->ODR |=  ( 1U << 3); // PB3
293             GPIOA->ODR |=  ( 1U << 0); // PA0
294             GPIOA->ODR &= ~( 1U << 1); // PA1
295             GPIOA->ODR &= ~( 1U << 4); // PA4
296             GPIOA->ODR &= ~( 1U << 5); // PA5
297             GPIOA->ODR |=  ( 1U << 12); // PA12
298             GPIOA->ODR |=  ( 1U << 11); // PA11
299             break;
300 
301         case 31: //L
302             GPIOB->ODR |=  ( 1U << 3); // PB3
303             GPIOA->ODR |=  ( 1U << 0); // PA0
304             GPIOA->ODR |=  ( 1U << 1); // PA1
305             GPIOA->ODR &= ~( 1U << 4); // PA4
306             GPIOA->ODR &= ~( 1U << 5); // PA5
307             GPIOA->ODR &= ~( 1U << 12); // PA12
308             GPIOA->ODR |=  ( 1U << 11); // PA11
309             break;
310 
```

```
311         case 32: //n
312             GPIOB->ODR |=  ( 1U << 3); // PB3
313             GPIOA->ODR |=  ( 1U << 0); // PA0
314             GPIOA->ODR &= ~( 1U << 1); // PA1
315             GPIOA->ODR |=  ( 1U << 4); // PA4
316             GPIOA->ODR &= ~( 1U << 5); // PA5
317             GPIOA->ODR |=  ( 1U << 12); // PA12
318             GPIOA->ODR &= ~( 1U << 11); // PA11
319             break;
320 
321         case 33: //D
322             GPIOB->ODR |=  ( 1U << 3); // PB3
323             GPIOA->ODR &= ~( 1U << 0); // PA0
324             GPIOA->ODR &= ~( 1U << 1); // PA1
325             GPIOA->ODR &= ~( 1U << 4); // PA4
326             GPIOA->ODR &= ~( 1U << 5); // PA5
327             GPIOA->ODR |=  ( 1U << 12); // PA12
328             GPIOA->ODR &= ~( 1U << 11); // PA11
329             break;
330 
331         case 34: // negative sign
332             GPIOB->ODR |= ( 1U << 3); // PB3
333             GPIOA->ODR |= ( 1U << 0); // PA0
334             GPIOA->ODR |= ( 1U << 1); // PA1
335             GPIOA->ODR |= ( 1U << 4); // PA4
336             GPIOA->ODR |= ( 1U << 5); // PA5
337             GPIOA->ODR |= ( 1U << 12); // PA12
338             GPIOA->ODR &= ~( 1U << 11); // PA11
339             break;
340 
341         case 35: // space
342             GPIOB->ODR |= ( 1U << 3); // PB3
343             GPIOA->ODR |= ( 1U << 0); // PA0
344             GPIOA->ODR |= ( 1U << 1); // PA1
345             GPIOA->ODR |= ( 1U << 4); // PA4
346             GPIOA->ODR |= ( 1U << 5); // PA5
347             GPIOA->ODR |= ( 1U << 12); // PA12
348             GPIOA->ODR |= ( 1U << 11); // PA11
349             break;
350         }
351 }
352 
353 
354 
355 
356 void utility_SSD(float var){
357 
358     int number = (int)var;
359 
360     float i = 0.0;
361 
362 
363 
364     number = (int)(var * i);
365 
366     int temp = number / 10;
367     Display.Digits[0] = (uint8_t)(number - (temp*10));
368 
369     temp = number / 100;
370     Display.Digits[1] = (uint8_t)((number - (temp * 100)) / 10);
371 
372     temp = number / 1000;
```

```
373     Display.Digits[2] = (uint8_t)((number - (temp * 1000)) / 100);
374
375     temp = number / 10000;
376     Display.Digits[3] = (uint8_t)((number - (temp * 10000)) / 1000);
377
378
379
380 }
381
382
```

```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];

}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 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
/*
 *
 *  Created on: Dec 29, 2020
 *      Author:MAKİF
 */

#include "eeprom.h"
#include "string.h"
#include "stdlib.h"

void read_ee(uint16_t regAddr, uint8_t *data, uint32_t num){
    uint8_t devAddr = 0x50;
    uint8_t arr[2];
    arr[0] = (uint8_t)(regAddr >> 8);
    arr[1] = (uint8_t)(regAddr & 0x00FF);
    write_general(devAddr, arr, 2);
    read_general(devAddr, data, num);
}

void write_ee(uint16_t regAddr, uint8_t *data, uint32_t num){
    uint8_t devAddr = 0x50;
    uint8_t arr[2];
    arr[0] = (uint8_t)(regAddr >> 8);
    arr[1] = (uint8_t)(regAddr & 0x00FF);
    uint8_t *ARR;
    ARR = (uint8_t*)malloc((num+2));
    memcpy(ARR, arr, 2);
    memcpy(ARR+2, data, num);
    write_general(devAddr, ARR, num+2);
    free(ARR);
}

```

```
1 /*
2  *
3  *
4  *  Created on: Dec 29, 2020
5  *      Author: MAKİF
6  */
7
8 #ifndef EEPROM_H_
9 #define EEPROM_H_
10
11 #include "main.h"
12 #include "I2C.h"
13
14 void read_ee(uint16_t regAddr, uint8_t *data, uint32_t num);
15 void write_ee(uint16_t regAddr, uint8_t *data, uint32_t num);
16
17 #endif /* EEPROM_H_ */
18
19
```

```c
/*
 *
 *  Created on: Dec 29, 2020
 *      Author: MAKİF
 */

#include "I2C.h"


void I2C1_IRQHandler(void){

}


void I2C_init_(void){

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

    //SCL
    //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);

    //SDA
    //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_general(uint8_t devAddr, uint8_t *data, uint32_t num){
```

```
63      while((I2C1->ISR & (1 << 15)));
64
65      I2C1->CR2 = 0;
66      I2C1->CR2 |= ((uint32_t)devAddr << 1); // slave address
67      I2C1->CR2 |= (num << 16); // Number of byte
68      I2C1->CR2 |= (1U << 25); // AUTOEND
69      I2C1->CR2 |= (1U << 13); // Generate Start
70
71
72      for (uint32_t i=0; i<num; ++i){
73          while(!(I2C1->ISR & (1 << 1))); //TXIS
74          I2C1->TXDR = (uint32_t)data[i];
75      }
76  }
77
78
79  void read_general(uint8_t devAddr, uint8_t *data, uint32_t num){
80
81      I2C1->CR2 = 0;
82      I2C1->CR2 |= ((uint32_t)devAddr << 1);
83      I2C1->CR2 |= (1U << 10); //READ mode
84      I2C1->CR2 |= (num << 16); //Number of bytes
85      I2C1->CR2 |= (1U << 15); //NACK
86      I2C1->CR2 |= (1U << 25); //AUTOEND
87      I2C1->CR2 |= (1U << 13); //Generate Start
88
89      for(uint32_t i=0; i<num; i++){
90          while(!(I2C1->ISR & (1 << 2))); // wait until RXNE =1
91          data[i] = (uint8_t)I2C1->RXDR;
92          }
93  }
94
95
```

```c
/*
 *
 *  Created on: Dec 29, 2020
 *      Author: MAKİF
 */

#ifndef I2C_H_
#define I2C_H_

#include "main.h"


void I2C_init_();
void read_general(uint8_t devAddr, uint8_t *data, uint32_t num);
void write_general(uint8_t devAddr, uint8_t *data, uint32_t num);


#endif /* I2C_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.
 */
#include "keypad.h"

/*to the reach delay func*/
extern void delay_ms(volatile unsigned int);
extern uint32_t k;
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


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

```
63          EXTI->RTSR1 |= (1U << 2);// 2th pin
64          EXTI->RTSR1 |= (1U << 8);// 8th pin
65
66
67          /* MASK*/
68          EXTI->IMR1 |= (1U << 9);// 9th pin
69          EXTI->IMR1 |= (1U << 0);// 0th pin
70          EXTI->IMR1 |= (1U << 2);// 2th pin
71          EXTI->IMR1 |= (1U << 8);// 8th pin
72
73
74          /*NVIC*/
75          NVIC_SetPriority(EXTI0_1_IRQn,0);
76          NVIC_EnableIRQ(EXTI0_1_IRQn);
77
78          NVIC_SetPriority(EXTI2_3_IRQn,0);
79          NVIC_EnableIRQ(EXTI2_3_IRQn);
80
81          NVIC_SetPriority(EXTI4_15_IRQn,0);
82          NVIC_EnableIRQ(EXTI4_15_IRQn);
83
84 }
85 /* interrut for PB0*/
86 void EXTI0_1_IRQHandler(void){
87     if (EXTI->RPR1 & (1U << 0)){// check if pending register equal 1
88
89          clear_rows_keypad();
90          /* make PA8 enable*/
91          GPIOA->ODR ^=( 1U << 8);
92          if ((GPIOB->IDR >> 0) & 1){//check if PB0 equal 1
93              /* #=(F) character*/
94              Keypad_data(15);
95
96          }
97          /*make PA8 disable*/
98          GPIOA->ODR ^=( 1U << 8); // PA8
99
100         /* make PB9 enable*/
101         GPIOB->ODR ^=( 1U << 9); // PB9
102         if ((GPIOB->IDR >> 0) & 1){
103             /* 9 character*/
104             Keypad_data(9);
105
106         }
107         /* make PB9 disable*/
108         GPIOB->ODR ^=( 1U << 9); // PB9
109
110         /* make PB5 enable*/
111         GPIOB->ODR ^=( 1U << 5); // PB5
112         if ((GPIOB->IDR >> 0) & 1){
113             /* 6 character*/
114             Keypad_data(6);
115
116         }
117         /* make PB5 disable*/
118         GPIOB->ODR ^=( 1U << 5); // PB5
119
120         /* make PB4 enable*/
121         GPIOB->ODR ^=( 1U << 4); // PB4
122         if ((GPIOB->IDR >> 0) & 1){
123             /* 3 character*/
124             Keypad_data(3);
```

```
125
126         }
127         /* make PB4 disable*/
128         GPIOB->ODR ^=( 1U << 4); // PB4
129
130
131         set_rows_keypad();
132         /*clear interrupt for clear pending register */
133         EXTI->RPR1 |= (1U << 0);
134     }
135 }
136
137 /* interrut for PB2*/
138 void EXTI2_3_IRQHandler(void){
139
140     if (EXTI->RPR1 & (1U << 2)){// check if pending register equal 1
141
142         clear_rows_keypad();
143         /*make PA8 enable*/
144         GPIOA->ODR ^=( 1U << 8); // PA8
145         if ((GPIOB ->IDR >> 2) & 1){//check if PB2 equal 1
146             /* D character*/
147             Keypad_data(13);
148
149         }
150         /*make PA8 disable*/
151         GPIOA->ODR ^=( 1U << 8); // PA8
152
153         /* make PB9 enable*/
154         GPIOB->ODR ^=( 1U << 9); // PB9
155         if ((GPIOB ->IDR >> 2) & 1){
156             /* C character*/
157             Keypad_data(12);
158
159         }
160         /* make PB9 disable*/
161         GPIOB->ODR ^=( 1U << 9); // PB9
162
163         /* make PB5 enable*/
164         GPIOB->ODR ^=( 1U << 5); // PB5
165         if ((GPIOB ->IDR >> 2) & 1){
166             /* B character*/
167             Keypad_data(11);
168
169         }
170         /* make PB5 disable*/
171         GPIOB->ODR ^=( 1U << 5); // PB5
172
173         /* make PB4 enable*/
174         GPIOB->ODR ^=( 1U << 4); // PB4
175         if ((GPIOB ->IDR >> 2) & 1){
176             /* A character*/
177             Keypad_data(10);
178
179         }
180         /* make PB4 disable*/
181         GPIOB->ODR ^=( 1U << 4); // PB4
182
183
184         set_rows_keypad();
185         /*clear interrupt for clear pending register */
186         EXTI->RPR1 |= (1U << 2);
```

```
187     }
188 }
189
190 /* interrut for PB8 and PA9*/
191 void EXTI4_15_IRQHandler(void){
192
193             /*interrut for PB8*/
194         if (EXTI->RPR1 & (1U << 8)){// check if pending register equal 1
195             clear_rows_keypad();
196             /*make PA8 enable*/
197             GPIOA->ODR ^=( 1U << 8); // PA8
198             if ((GPIOB ->IDR >> 8) & 1){//check if PB8 equal 1
199                 /* *(E) character*/
200                 Keypad_data(14);
201
202             }
203             /*make PA8 disable*/
204             GPIOA->ODR ^=( 1U << 8); // PA8
205
206             /* make PB9 enable*/
207             GPIOB->ODR ^=( 1U << 9); // PB9
208             if ((GPIOB ->IDR >> 8) & 1){
209                 /* 7 character*/
210                 Keypad_data(7);
211
212             }
213             /* make PB9 disable*/
214             GPIOB->ODR ^=( 1U << 9); // PB9
215
216             /* make PB5 enable*/
217             GPIOB->ODR ^=( 1U << 5); // PB5
218             if ((GPIOB ->IDR >> 8) & 1){
219                 /* 4 character*/
220                 Keypad_data(4);
221
222             }
223             /* make PB5 disable*/
224             GPIOB->ODR ^=( 1U << 5); // PB5
225
226             /* make PB4 enable*/
227             GPIOB->ODR ^=( 1U << 4); // PB4
228             if ((GPIOB ->IDR >> 8) & 1){
229                 /* 1 character*/
230                 Keypad_data(1);
231
232             }
233             /* make PB4 disable*/
234             GPIOB->ODR ^=( 1U << 4); // PB4
235
236
237             set_rows_keypad();
238             /*clear interrupt for clear pending register */
239             EXTI->RPR1 |= (1U << 8);
240         }
241
242         /*interrut for PA9*/
243         if (EXTI->RPR1 & (1U << 9)){// check if pending register equal 1
244             clear_rows_keypad();
245             /*make PA8 enable*/
246             GPIOA->ODR ^=( 1U << 8); //check if PA8 equal 1
247             if ((GPIOA ->IDR >> 9) & 1){
248                 /* 0 character*/
```

```
249                 Keypad_data(0);
250 
251             }
252             /*make PA8 disable*/
253             GPIOA->ODR ^=( 1U << 8); // PA8
254 
255             /* make PB9 enable*/
256             GPIOB->ODR ^=( 1U << 9); // PB9
257             if ((GPIOA ->IDR >> 9) & 1){
258                 /* 8 character*/
259                 Keypad_data(8);
260 
261             }
262             /* make PB9 disable*/
263             GPIOB->ODR ^=( 1U << 9); // PB9
264 
265             /* make PB5 enable*/
266             GPIOB->ODR ^=( 1U << 5); // PB5
267             if ((GPIOA ->IDR >> 9) & 1){
268                 /* 5 character*/
269                 Keypad_data(5);
270 
271             }
272             /* make PB5 disable*/
273             GPIOB->ODR ^=( 1U << 5); // PB5
274 
275             /* make PB4 enable*/
276             GPIOB->ODR ^=( 1U << 4); // PB4
277             if ((GPIOA ->IDR >> 9) & 1){
278                 /* 2 character*/
279                 Keypad_data(2);
280 
281             }
282             /* make PB4 disable*/
283             GPIOB->ODR ^=( 1U << 4); // PB4
284 
285 
286             set_rows_keypad();
287 
288             /*clear interrupt for clear pending register */
289             EXTI->RPR1 |= (1U << 9);
290         }
291 
292 }
293 
294 
295 void clear_rows_keypad(void){
296         /*clearing the rows here*/
297         GPIOA->ODR &= ~(1U << 8);//PA8
298         GPIOB->ODR &= ~(1U << 9);//PB9
299         GPIOB->ODR &= ~(1U << 5);//PB5
300         GPIOB->ODR &= ~(1U << 4);//PB4
301 }
302 
303 void set_rows_keypad(void){
304         /*seting the rows here*/
305         GPIOA->ODR |= (1U << 8);//PA8
306         GPIOB->ODR |= (1U << 9);//PB9
307         GPIOB->ODR |= (1U << 5);//PB5
308         GPIOB->ODR |= (1U << 4);//PB4
309 
310 }
```

```
311
312 void Keypad_data(uint8_t input){
313
314     switch(input){
315
316             case 0:
317     //          delete_selected_track();
318
319             break;
320
321             case 1:
322     //          play/resume(); //selected
323                 k=1;
324
325                 break;
326             case 2:
327     //          pause();
328                 k=3;
329
330             case 10:
331     //          record();
332                 k=0;
333
334                 break;
335
336         }
337 }
338
339
340
```

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

#ifndef KEYPAD_H_
#define KEYPAD_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_ */

```

```c
/*
 * main.c
 *
 *  Created on: Jan 10, 2021
 *      Author:
 */


#include "bsp.h"

int main(void) {


    BSP_init();

    while(1) {

    }

    return 0;
}

```

```
1 /*
2  * SSD.c
3  *
4  *  Created on: Dec 19, 2020
5  *      Author:
6  */
7
8 #ifndef MAIN_H_
9 #define MAIN_H_
10
11 #include "stm32g0xx.h"
12
13
14 #endif /* MAIN_H_ */
15
```