GEBZE TECHNİCAL UNIVERTİSY

ELEC 335

PROJECT - 03

REPORT

DİGİTAL VOİCE RECORDER

BERKAY TÜRK

171024024

INTRODUCTION:

Our aim in this Project is to create a digital voice recorder.

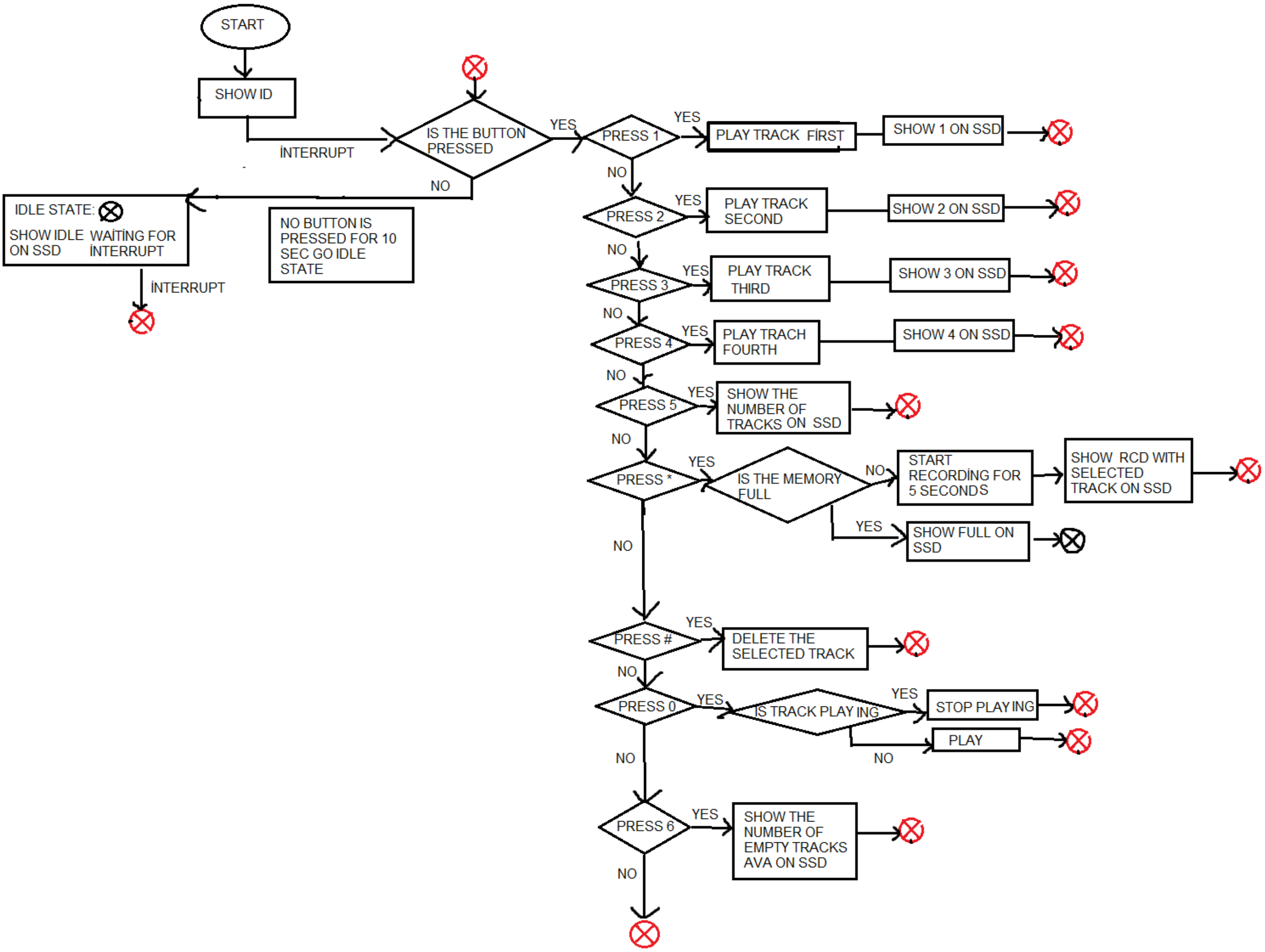
Detailed Requirements :

- Written in C
- 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 mind when wiring the bus will require pull-up resistors on both lines, and each of these devices need different 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 different 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.
- 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.
- If no button is pressed for 10 seconds, the device should go back to IDLE state.
- 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)

I first created a flowchar in this direction and I dive into small task in flowchart and Finally I combine the tasks.



Flowchart

TASK 1: (+)

Connect one 4xSSD to the board and turn on one part of a segment and I knew how it all turned on and off .My SSD is common katot .I make figure 1.

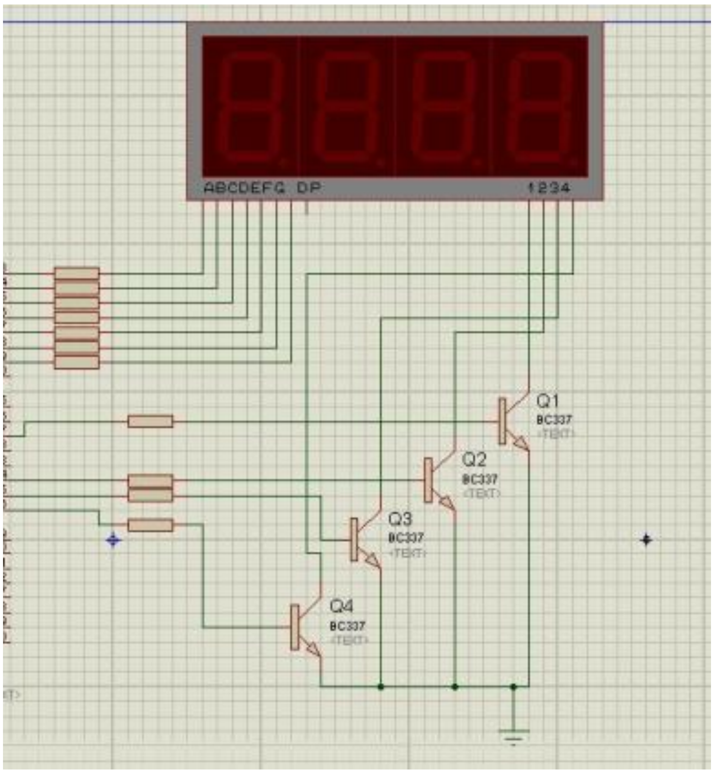


Figure 1.

TASK 2: (+)

I connect to Keypad the way I learned from the applications lesson and I know connect leds and button and I make figure 2.

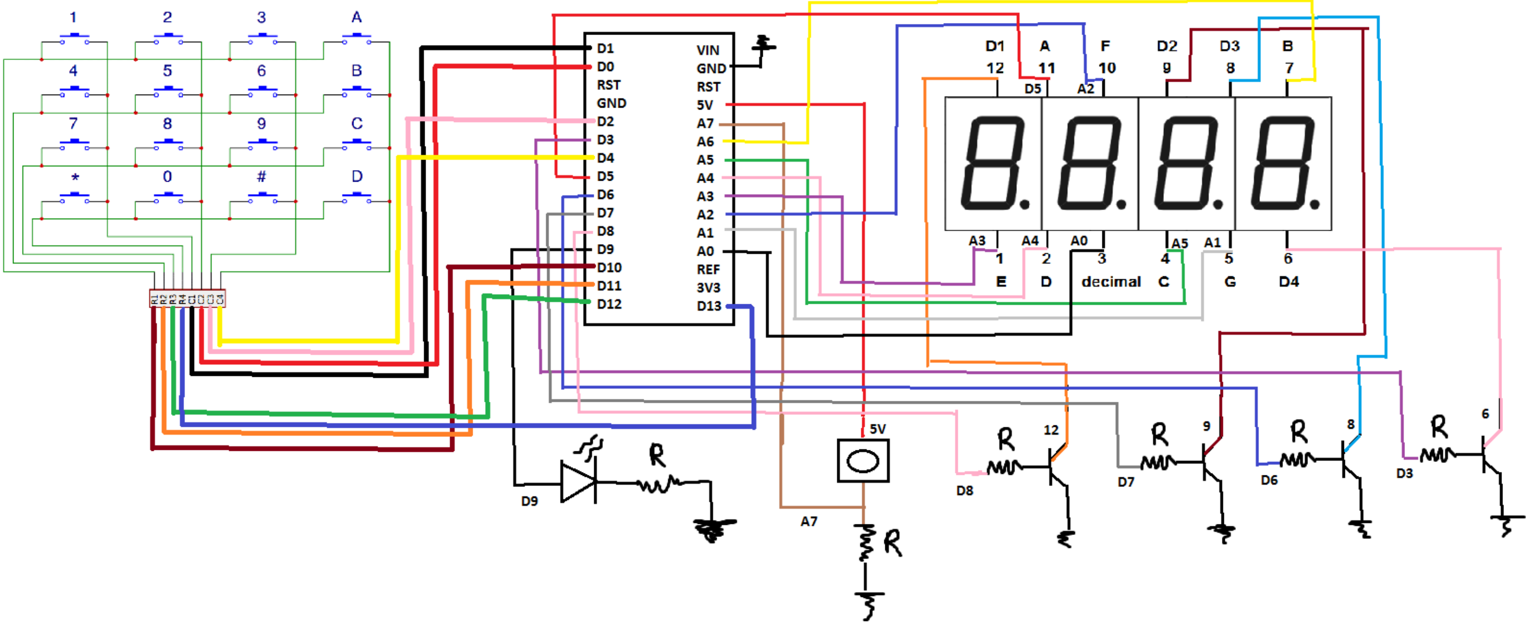


Figure 2.

TASK 3: (+)

I removed some unused pins to make space in my board.And I learned how to connect the speaker,microphone and EEPROMs and I make figure 3.

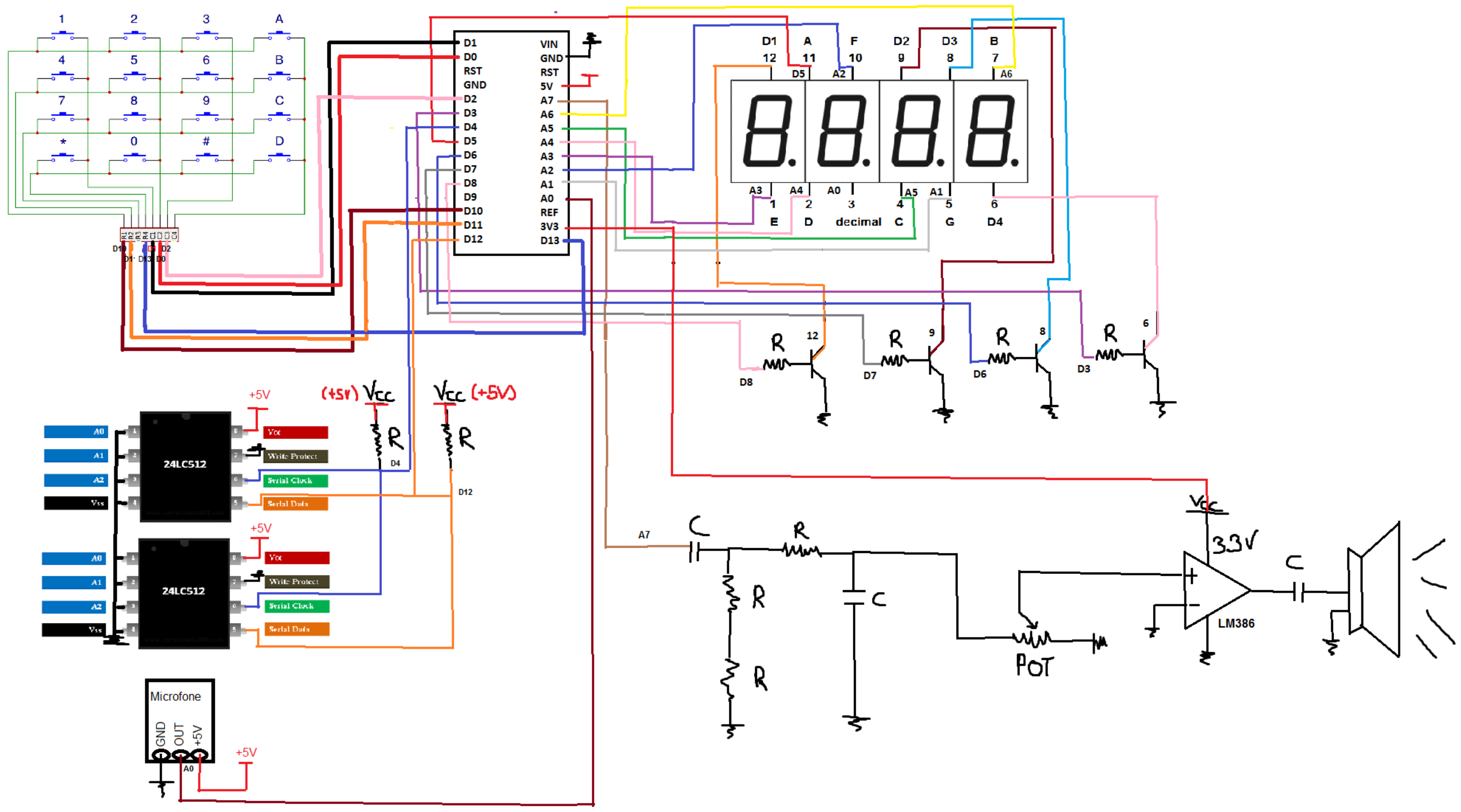


Figure 3. Connection Diagram

TASK 4: (+)

My flowchart is too long and I divide small piece. I knew how it's done to show our school number and i learned to interrupt to keypad and write it.

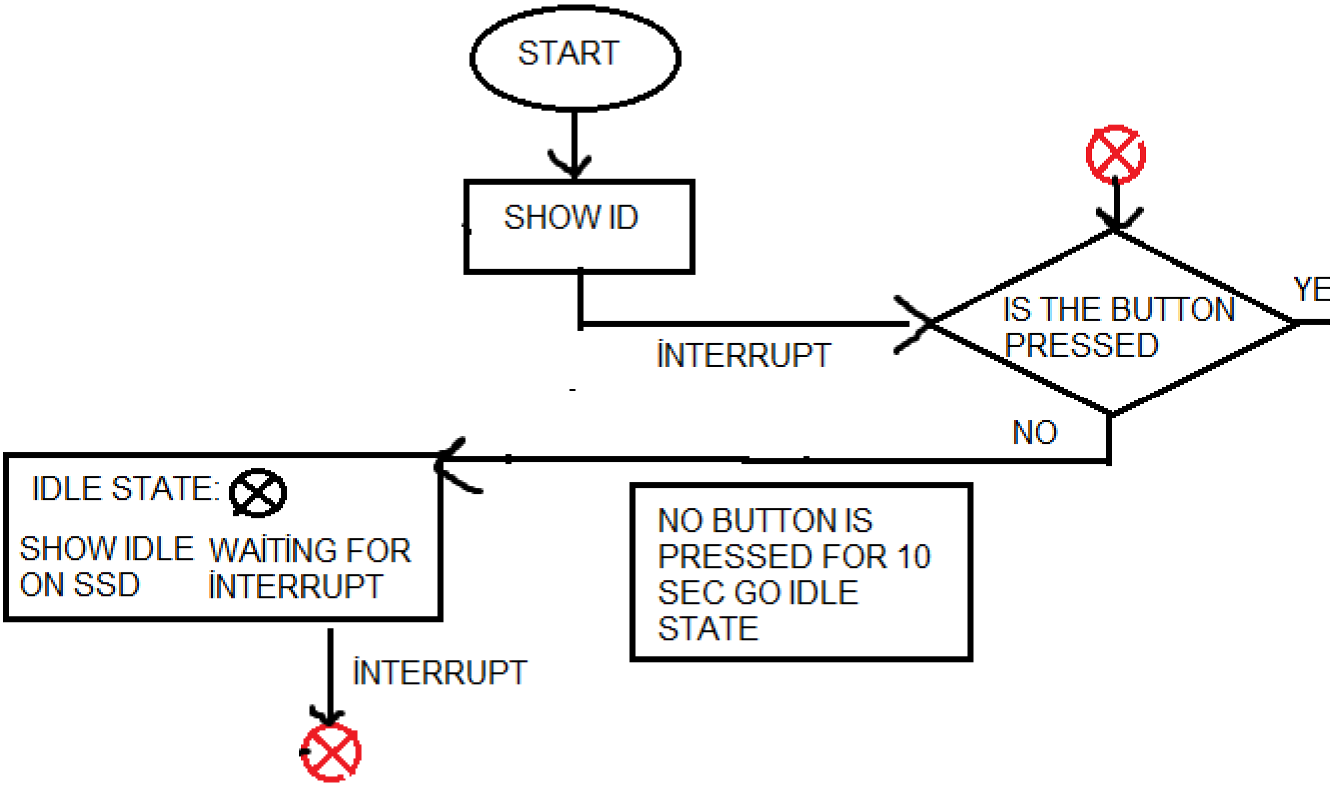


Figure 4.

TASK 5: (-)

After I work figure 5.But I couldn't do how to save the record.

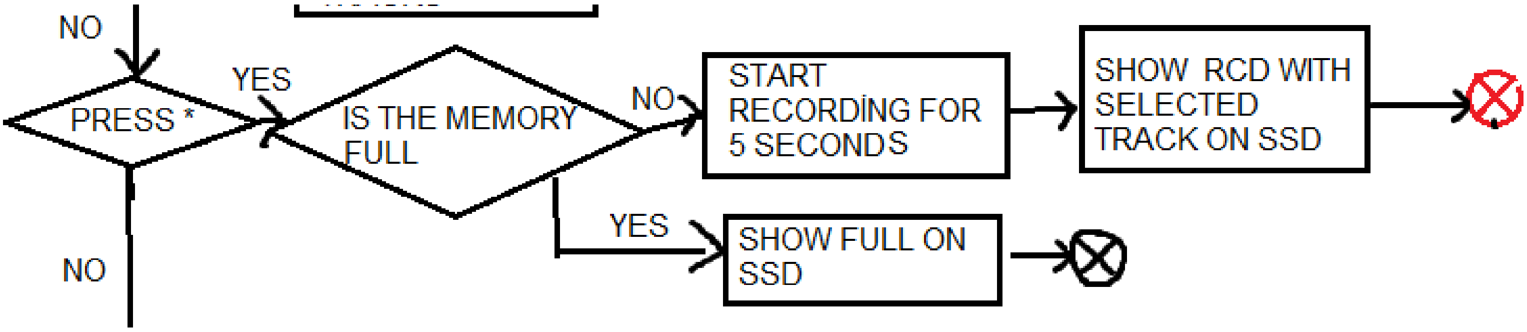

NO
PRESS *
YES
IS THE MEMORY FULL
NO
START RECORDİNG FOR 5 SECONDS
SHOW RCD WITH SELECTED TRACK ON SSD
YES
SHOW FULL ON SSD
NO

Figure 5.

TASK 6: (-)

And I work figure 6. I first thought it simple I tried to make a sound from the speaker I failed this too. There was no sound for some reason

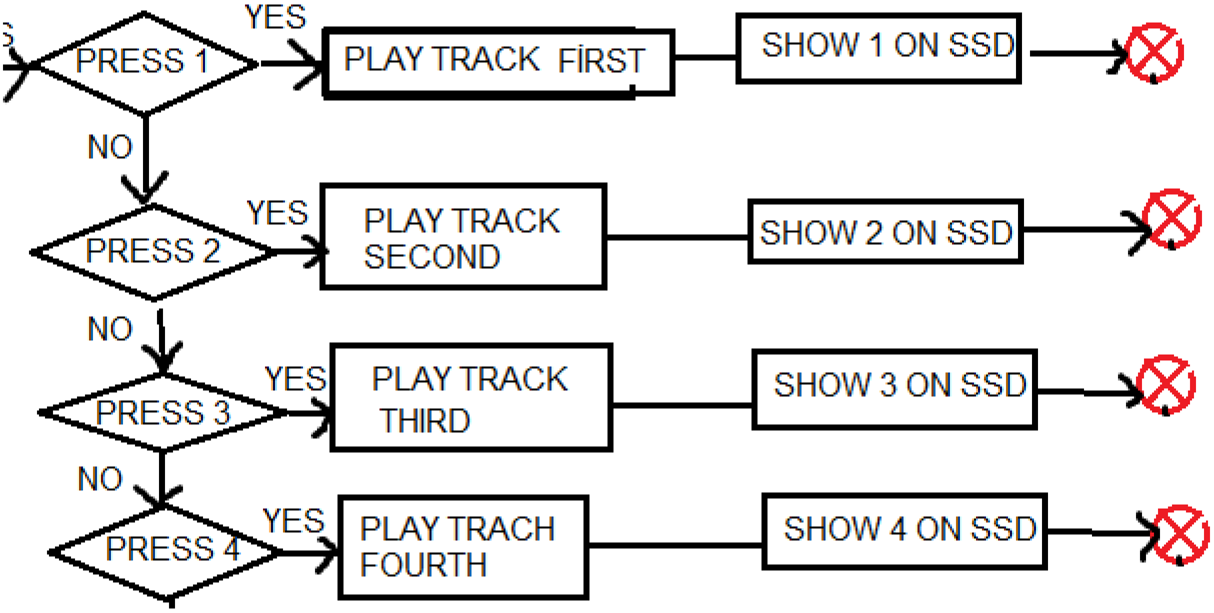


Figure 6.

TASK 7: (-)

And same way I work figure 7 I had difficulty in this part because I couldn't record

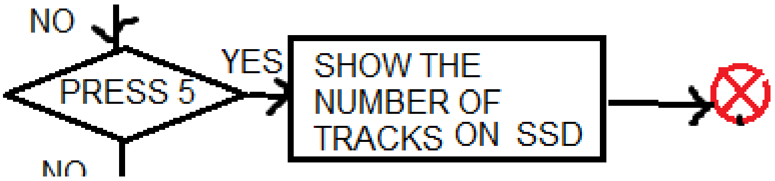


Figure 6.

TASK 8: (-)

And same way I work figure 8. I had difficulty in this part because I couldn't record.

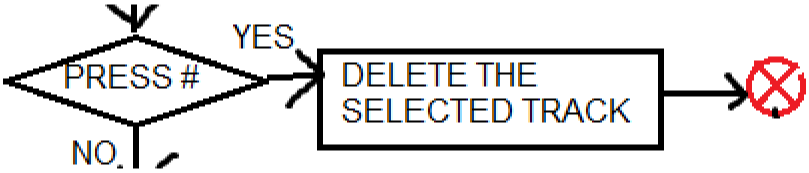


Figure 8.

TASK 9: (-)

And same way I work figure 9. I could not do this part because I have problems with EEPROMs

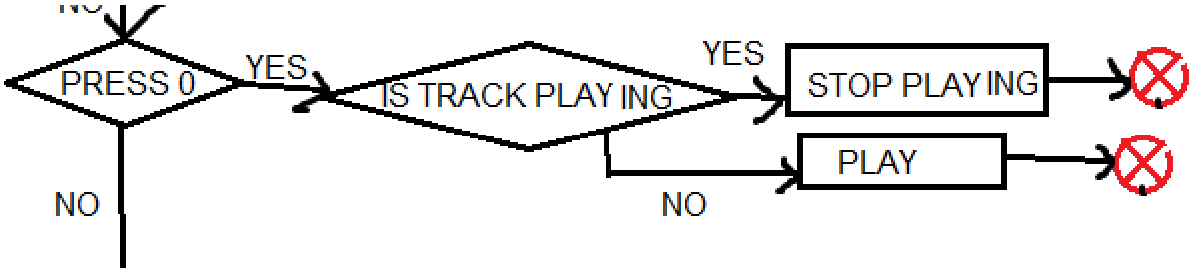


Figure 9.

TASK 10: (-)

And same way I work figure 10. I had difficulty in this part because I couldn't record.

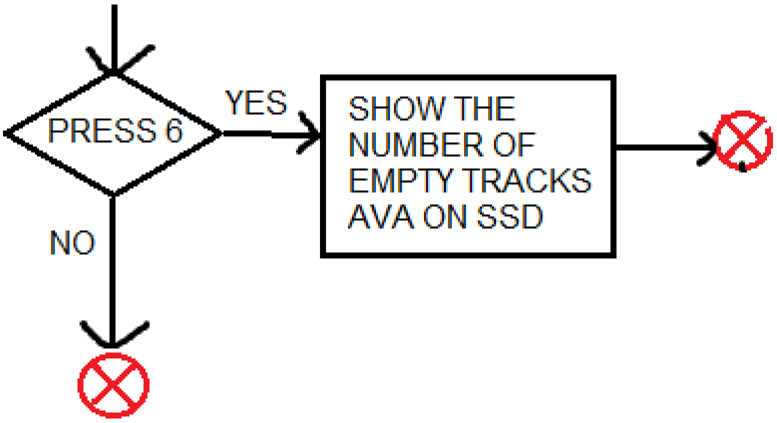
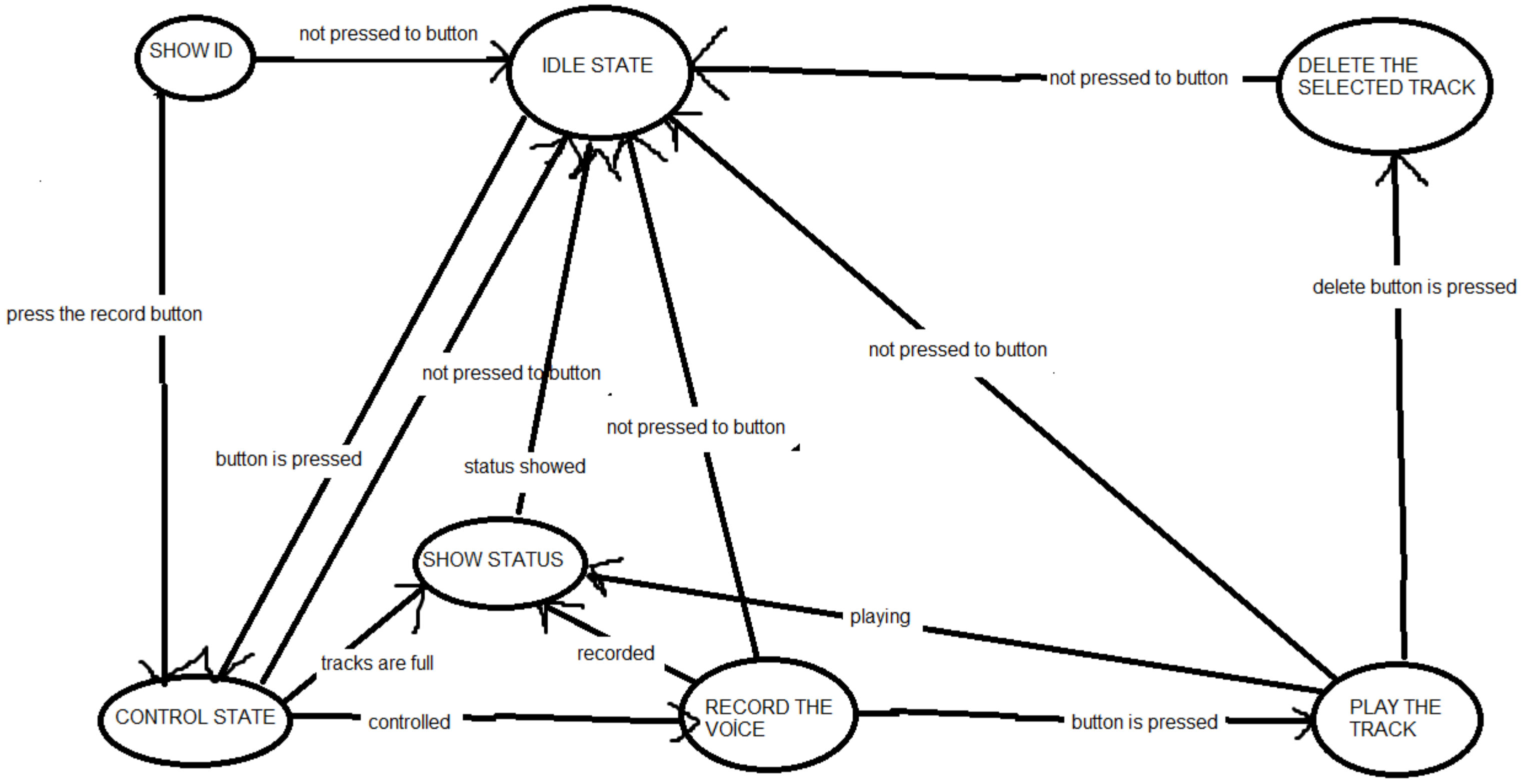
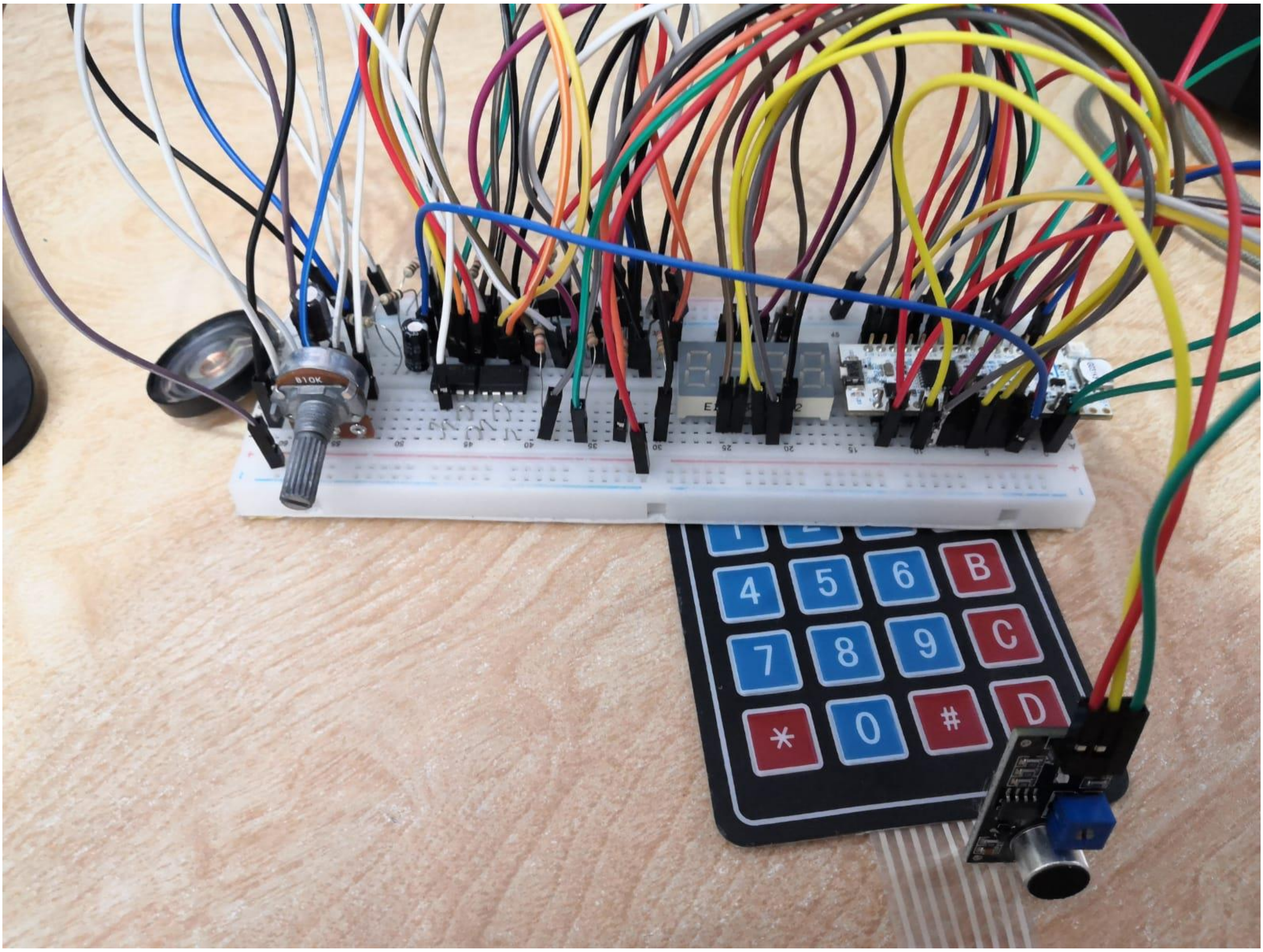


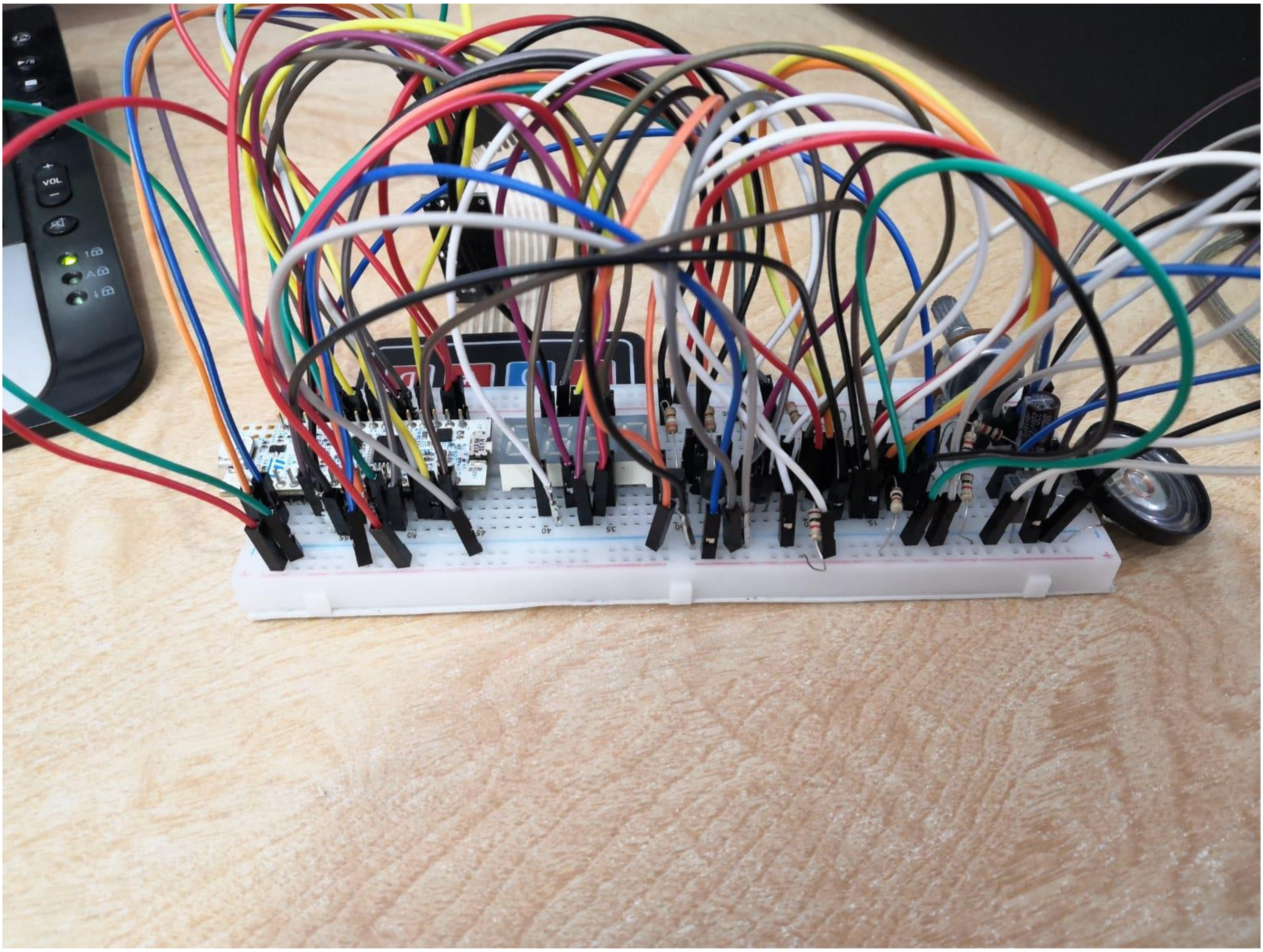
Figure 10.



State transition diagram for digital voice recorder



Front



Back

PART LİST:

NUCLEO-G031K8	**X1**	**110TL**
JUMPER CABLE	**X50**	**10TL**
RESİSTANCE 470Ω	**X6**	**1TL**
RESİSTANCE 1kΩ	**X3**	**0.5TL**
POT(10k Ω)		**3 TL**
CAPASİTANCE (1uF,220uF,47nF)		**2 TL**
4XSEVEN SEGMENT	**X1**	**7TL**
4x4 KEYPAD	**X1**	**10TL**
MİCROPHONE	**X1**	**15TL**
SPEAKER	**X1**	**5TL**
TRANSİSTOR	**X4**	**1TL**
BUTTON	**X1**	**0.25TL**
LED	**X1**	**0.25TL**
SUM		**165TL**

MATHEMATİCAL WORKS:

Set System Clock

I want system clock as $fSYSCLK$=64 MHz, so $PLLM$=1, N=8, R=2, Q=2.

$$\mathrm{fSYSCLK} = \frac{\mathrm{HSIRC} * \mathrm{N}}{\mathrm{PLLM} * \mathrm{R} * \mathrm{Q}} = \frac{16 * 8}{1 * 2 * 2\mathrm{MHz}} = 64\ \mathrm{MHz}$$

Set TIM1 as PWM Out (PA7)

$$\mathrm{fPWM} = \frac{\mathrm{fSYSCLK}}{\mathrm{TIM1Period} * (1 + \mathrm{TIM1\ Prescaler})} \gg \mathrm{fS}$$

DAC for sampling frequency fS=8kHz I can pick $TIM1Period$=255, $TIM1\ Prescaler$=0

$$fPWM{=}250kHz{\gg}8kHz$$

This $fPWM{\gg}fS$ constraint, helps us to achieve less noisy output signal.

Speaker Power Constraints

I have 8Ω, $0.5\mathrm{W}$ speaker. I have a max voltage constraint to use this speaker without burning it.

$$P=\frac{\mathrm{V}^2}{\mathrm{R}}\Rightarrow 0.5=\frac{|VSIG|\hat{}2_{\mathrm{MAX}}}{8}\Rightarrow |VSIG|MAX=2$$

I will design amplifier due to $|V_{SIG}|_{MAX}=2$ constraint .

CONCLUSİON:

As a result, I leard to how speaker and microfone is connected the board and how is interrupt . I learned how to generate pwm signal. I learned some information about EEPROMs.

I tried to do a digital voice recorder but I could not reach a certain result.

This project is open to improve because more can be recorded.

The biggest challenge is EEPORMs because I could not figure out the logic of these tools.

VİDEO LİNK:

Code explanation:

https://youtu.be/Odn4-gQ8ni4

REFERANCES:

The_Definitive_Guide_to_ARM_CortexM0_M0+ *Second Edition* Joseph Yiu

RM0444 Reference manual

https://elektrokod.wordpress.com/2013/12/09/7-segment-display-sayici-uygulamasi/

https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet

CODE:

Main.c

```c
/*
 * main.c
 *
 * author: Berkay Türk 171024024
 *
 * description: In this project, make a digital voice recorder.
 *   G031K8 Nucleo board.
 */

#include "stm32g0xx.h"
#include "time.h"
#include "stdio.h"
#include "bsp.h"


#define LEDDELAY 1600000U

int main(void) {

      BSP_System_init();
      init_adc() ;
      init_timer1();
    init_I2C();

for(;;){

      Keypad_enable();

  }
    return 0;
}
```

Bsp.h

```
#ifndef BSP_H_
#define BSP_H_
#include "stm32g0xx.h"

/* Common API functions for nucleo */

void delay_ms(uint32_t);
void delay(volatile unsigned int);


void showID();
void BSP_UART_init(uint32_t);
void printChar(uint8_t);
void BSP_System_init();

void init_adc() ;
unsigned int ADC_start(void);
void init_timer1() ;
void EXTI4_15_IRQHandler ();
// LED related functions
void Keypad_enable();
void BSP_led_init();
void BSP_led_set();
void BSP_led_clear();
void BSP_led_toggle();

void setSSD(int x , int y);
void SwitchSSD(int x);

// Button related functions
void BSP_button_init();
int BSP_button_read();

void IDLE_state();
void clearSSD();
void setRowsKeypad();
void clearRowsKeypad();
#endif
```

Bsp.c

```c
#include "stm32g0xx.h"
#include "bsp.h"
#include "math.h"
#include "time.h"

#define LC512_ADDRESS 0X68
#define LC512_ADDRESS 0X65

#define LC512_WHO_AM_I 0X75
#define LC512_PWR_MGMT_1 0X6B


static volatile  uint32_t tick = 0;
int t=0;
static volatile int analogvalue;

void BSP_led_init(void) {

       /* Enable GPIOA clock */  /* Enable GPIOB clock */
           RCC->IOPENR |= (3U << 0);

                 /* setup PA(0,1,4,5,6,8,9,11,12) for seven segment A,B,C,D,E,F,G,DH for bits in MODER
*/
                        GPIOA->MODER &= ~(0x3CF3F0F);
                        GPIOA->MODER |= (0x1451505);
                     /* setup PB(0,1,2,8) for seven segment D4,D3,D2,D1 for in MODER */
                        GPIOB->MODER &= ~(0x3003F);
                        GPIOB->MODER |= (0x10015);

}

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

void delay_ms(uint32_t s) {
    tick = s;
    while(tick);
}

void SysTick_Handler(void) {

      if(tick > 0){

            --tick;
      }

}
void init_timer1() {

      RCC->APBENR1 |= (1U << 1); // enable TIM3 module clock

      TIM3->CCR3 = 0; // Zero out the control register just in case
      TIM3->CR1 |= (1 << 7); // ARPE
      TIM3->CNT = 0; // Zero out counter
    TIM3->CCMR2 |= (111 << 16); // PWM Mode1
    TIM3->CCER |= (00 << 0); // capture / compare output

    /// 1  second interrupt
      TIM3->PSC = 999;
      TIM3->ARR = 1600;

      TIM3->DIER |= (1 << 0);  // update interrupt enable
      TIM3->CR1  |= (1 << 0);  // TIM1 Enable

    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 1);
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
}
```

```
//Timer Handler
void TIM3_IRQHandler(){
      TIM3->ARR =160;                        //Adjusting auto-reload to change the delay time with
each press
      TIM3->SR &=  ~(1U << 0);  //Reset Timer

//If not pressed or not counting do nothing
}

void init_timer3(){ //Set to create exception each 0.0001 second
      RCC->APBENR1 |= (1U << 1);                        //Enabling TIM3
      TIM3->CR1 = 0;                                          //RESET TIM3_CR1 register
      TIM3->CR1 |= (1 << 7);                            //AUTO RELOAD ENABLED
      TIM3->DIER |=(1 << 0);                            //UPDATE INTERRUPT ENABLED
      TIM3->CNT = 0 ;                                         //RESET COUNTER

      TIM3->PSC = 99;                                         //PRESCALER SET to 9
      TIM3->ARR =160;                                         //AUTORELOAD VALUE
(PSC+1*ARR)/SystemCoreClock=0.0001
      TIM3->CR1 |= (1 << 0);                            //Counter enabled

      NVIC_SetPriority(TIM3_IRQn , 4);       //Set to the lowest priority level
      NVIC_EnableIRQ(TIM3_IRQn);                        //Enable interrupt
}

void TIM1_BRK_UP_TRG_COM_IRQHandler(void) {
    analogvalue = (int)ADC_start();
    double buffer = tick;
    for(;tick-buffer>1800;);
    if(analogvalue>2200){
    read_write_data();
    TIM1->SR &= ~(1U << 0);
    }
}
void EXTI4_15_IRQHandler(void) {   //INTERRUPT function
      clearSSD();
         //Small delay introduced to prevent bouincing
          delay(200);
          EXTI->RPR1 |= (1U << 5);    //Set hardware raised flag to zero by software

      if((EXTI->RPR1 >>6) & 1  ){/* Interrupt from PB6 */

      clearRowsKeypad();

      GPIOB->ODR ^= (1U << 9);  // PB9
      if((GPIOB->IDR >> 6) & 1 ){//'1'
            setSSD(1,3);
            read_write_data();

            //play track 1 and show 1 on SSD
            }

      GPIOB->ODR ^= (1U << 9);

      GPIOB->ODR ^= (1U << 5);  // PB5
      if((GPIOB->IDR >> 6) & 1 ){//'4'
            setSSD(4,3);
            //play track 4 and show 4 on SSD
      }

       GPIOB->ODR ^= (1U << 5);

       GPIOB->ODR ^= (1U << 4);  // PB4
       if((GPIOB->IDR >> 6) & 1 ){//'7'

             //NOTHING
      }
       GPIOB->ODR ^= (1U << 4);
       GPIOB->ODR ^= (1U << 3);  // PB3
       if((GPIOB->IDR >> 6) & 1 ){//*
             TIM1_BRK_UP_TRG_COM_IRQHandler();

             //start recording tracks
      }
       GPIOB->ODR ^= (1U << 3);

        EXTI->RPR1 |= (1U << 6);//Clear interrupt flag
        setRowsKeypad();

}
      if((EXTI->RPR1 >>7) & 1 ){/* Interrupt from PB7 */

      clearRowsKeypad();

      GPIOB->ODR ^= (1U << 9);  // PB9
      if((GPIOB->IDR >> 7) & 1 ){//'2'

            //play track 2 and show 2 on SSD
      }

      GPIOB->ODR ^= (1U << 9);

      GPIOB->ODR ^= (1U << 5);  // PB5
      if((GPIOB->IDR >> 7) & 1 ){//'5'

            //show the number of tracks on SSD
      }

       GPIOB->ODR ^= (1U << 5);

       GPIOB->ODR ^= (1U << 4);  // PB4
       if((GPIOB->IDR >> 7) & 1 ){//'8'

             //NOTHING
      }

       GPIOB->ODR ^= (1U << 4);

       GPIOB->ODR ^= (1U << 3);  // PB3
       if((GPIOB->IDR >> 7) & 1 ){//'0'

             //stop or play the track
      }
       GPIOB->ODR ^= (1U << 3);

       EXTI->RPR1 |= (1U << 7); //Clear interrupt flag
              setRowsKeypad();
       }


      if((EXTI->RPR1 >> 15) & 1 ){/* Interrupt from PA15 */

      clearRowsKeypad();

      GPIOB->ODR ^= (1U << 9);  // PB9
      if((GPIOA->IDR >> 15) & 1 ){//'3'

            //play track 3 and show 3 on SSD
      }
       GPIOB->ODR ^= (1U << 9);

      GPIOB->ODR ^= (1U << 5);  // PB5
      if((GPIOA->IDR >> 15) & 1 ){//'6'

            //show empty track on SSD
      }
```

```
       GPIOB->ODR ^= (1U << 5);

       GPIOB->ODR ^= (1U << 4);  // PB4
       if((GPIOA->IDR >> 15) & 1 ){//'9'

              //NOTHING
      }

       GPIOB->ODR ^= (1U << 4);

       GPIOB->ODR ^= (1U << 3);  // PB3
       if((GPIOA->IDR >> 15) & 1 ){//#

        //delete selected track
      }
       GPIOB->ODR ^= (1U << 3);

       EXTI->RPR1 |= (1U << 15);//Clear interrupt flag
               setRowsKeypad();
        }

      if((EXTI->RPR1 >> 10) & 1 ){/* Interrupt from PA10 */

      clearRowsKeypad();

      GPIOB->ODR ^= (1U << 9);  // PB9
      if((GPIOA->IDR >> 10) & 1 ){//A

             //NOTHING
      }

      GPIOB->ODR ^= (1U << 9);

      GPIOB->ODR ^= (1U << 5);  // PB5
      if((GPIOA->IDR >> 10) & 1 ){//B

             //NOTHING
      }

       GPIOB->ODR ^= (1U << 5);

       GPIOB->ODR ^= (1U << 4);  // PB4
       if((GPIOA->IDR >> 10) & 1 ){//C

              //NOTHING
      }

       GPIOB->ODR ^= (1U << 4);

       GPIOB->ODR ^= (1U << 3);  // PB3
       if((GPIOA->IDR >> 10) & 1 ){//D

              //NOTHING
      }
       GPIOB->ODR ^= (1U << 3);

       EXTI->RPR1 |= (1U << 10);//Clear interrupt flag
               setRowsKeypad();
        }

 delay(800000);//wait 1 sec because interrups go same

  }

void clearSSD(void) {

      /* Set all output connected to SSD (clear SSD)*/
      GPIOA->BRR |= (0x1A72);

}
```

```
void showNumber() {

  for (unsigned int retTime = time(0) + 2000; time(0) < retTime; retTime--){   // Loop until it arrives.

      showID();    //My school ID show and loop

       if(retTime == 0)//wait 10 sec and no press button go to clear SSD
         break;
  }

       IDLE_state();

}
void IDLE_state(){
       clearSSD();//off SSD
       while(1){
           //wait here until the press button
       }

}

void Keypad_enable(){

/*   Setup Output pins (rows) */

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


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


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


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


        /*   Setup Input pins (Columns)   */

        GPIOB->MODER &= ~(3U << 2*6);  /// PB6 is input
        GPIOB->PUPDR |= (2U << 2*6);   /// Pull-Down mode


        GPIOB->MODER &= ~(3U << 2*7);  /// PB7 is input
        GPIOB->PUPDR |= (2U << 2*7);    /// Pull-Down mode


        GPIOA->MODER &= ~(3U << 2*15);  /// PA15 is input
        GPIOA->PUPDR |= (2U << 2*15);    /// Pull-Down mode


        GPIOA->MODER &= ~(3U << 2*10);  /// PA10 is input
        GPIOA->PUPDR |= (2U << 2*10);    /// Pull-Down mode
           /* Setup interrupts for inputs */
        EXTI->EXTICR[1] |= (1U << 8*2);   // PB6
        EXTI->EXTICR[1] |= (1U << 8*3);   // PB7
        EXTI->EXTICR[3] |= (0U << 8*3);   // PA15
        EXTI->EXTICR[2] |= (0U << 8*2);   // PA10

        /* RISING Edge*/
        EXTI->RTSR1 |= (1U << 6);      // 6th pin
        EXTI->RTSR1 |= (1U << 7);      // 7th pin
        EXTI->RTSR1 |= (1U << 15);      // 15th pin
        EXTI->RTSR1 |= (1U << 10);      // 10th pin
        /* MASK*/
        EXTI->IMR1 |= (1U << 6);
        EXTI->IMR1 |= (1U << 7);
        EXTI->IMR1 |= (1U << 15);
        EXTI->IMR1 |= (1U << 10);

        /*NVIC */

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


           /* Setup all rows*/
       GPIOB->ODR |= (1U << 9);   /// PB9
       GPIOB->ODR |= (1U << 5);   /// PB5
       GPIOB->ODR |= (1U << 4);   /// PB4
       GPIOB->ODR |= (1U << 3);   /// PB3

       clearSSD();//turn off SSD
       while(1){
             if(t==0){ // start value t=0 must be in
            showNumber();    // show School number  wait here
              }

    }
}

void showID(){ //My school ID show
    setSSD(1 , 3);//1
    delay(1600);//delay ms
    setSSD(7 , 2);//7
    delay(1600);//delay ms
    setSSD(2 , 1);//2
    delay(1600);//delay ms
    setSSD(4 , 0);//4
    delay(1600);//delay ms
}

void SwitchSSD(int x) {


      switch (x)
             {

             case 0://'D'

                    /* turn on led connected to A,B,C,D,E,F in ODR*/
                   GPIOA->ODR |= (0x1A70);
                   /* turn off led connected to G in ODR*/
                   GPIOA->BRR |= (0x2);
                   break;

             case 1:
                    /* turn on led connected to B,C in ODR*/
                   GPIOA->ODR |= (0x840);
                   /* turn off led connected to A,D,E,F,G in ODR*/
                   GPIOA->BRR |= (0x1232);
                   break;
             case 2:
                    /* turn on led connected to A,B,D,E,G in ODR*/
                   GPIOA->ODR |= (0x1262);
                   /* turn off led connected to C,F in ODR*/
                   GPIOA->BRR |= (0x810);
                   break;
```

```
     case 3:
            /* turn on led connected to A,B,C,D,G in ODR*/
           GPIOA->ODR |= (0x1A42);
           /* turn off led connected to E,F in ODR*/
           GPIOA->BRR |= (0x30);
           break;
     case 4:
            /* turn on led connected to B,C,G,F in ODR*/
           GPIOA->ODR |= (0x852);
           /* turn off led connected to A,D,E in ODR*/
           GPIOA->BRR |= (0x1220);
           break;

     case 5:
            /* turn on led connected to A,C,D,F,G in ODR*/
           GPIOA->ODR |= (0x1A12);
           /* turn off led connected to B,E in ODR*/
           GPIOA->BRR |= (0x60);
           break;

     case 6:
            /* turn on led connected to A,B,C,D,E,F,G in ODR*/
           GPIOA->ODR |= (0x1A32);
           /* turn off led connected to B in ODR*/
           GPIOA->BRR |= (0x40);
           break;
     case 7:
            /* turn on led connected to A,B,C in ODR*/
           GPIOA->ODR |= (0xA40);
           /* turn off led connected to D,E,F,G in ODR*/
           GPIOA->BRR |= (0x1032);
           break;
     case 8://'B'
            /* turn on led connected to all in ODR*/
           GPIOA->ODR |= (0x1A72);
           break;

     case 9:
            /* turn on led connected to A,B,C,D,F,G in ODR*/
           GPIOA->ODR |= (0x1A52);
           /* turn off led connected to E in ODR*/
           GPIOA->BRR |= (0x20);
           break;
     case 10://'A'
            /* turn on led connected to A,B,C,F,E,G in ODR*/
           GPIOA->ODR |= (0xA72);
           /* turn off led connected to D in ODR*/
           GPIOA->BRR |= (0x1000);
    break;
     case 11://'V'
           /* turn on led connected to B,F,G in ODR*/
           GPIOA->ODR |= (0x52);
           /* turn off led connected to A,D,E,C in ODR*/
           GPIOA->BRR |= (0x1A20);
    break;

     case 12://'R'
           /* turn on led connected to A,D,E,B,F,G in ODR*/
           GPIOA->ODR |= (0x1272);
           /* turn off led connected to C in ODR*/
           GPIOA->BRR |= (0x800);
    break;
     case 13://'C'
           /* turn on led connected to A,D,E,F in ODR*/
           GPIOA->ODR |= (0x1230);
           /* turn off led connected to B,C,G in ODR*/
           GPIOA->BRR |= (0x842);
    break;
```

```
             case 14://'P'
                   /* turn on led connected to A,B,G,E,F in ODR*/
                   GPIOA->ODR |= (0x272);
                   /* turn off led connected to D,C in ODR*/
                   GPIOA->BRR |= (0x1800);
            break;
             case 15://'L'
                   /* turn on led connected to D,E,F in ODR*/
                   GPIOA->ODR |= (0x1030);
                   /* turn off led connected to A,B,G,C in ODR*/
                   GPIOA->BRR |= (0xA42);
            break;
             }
}
void setSSD(int x , int y) { // x is the number led(0 , 1)  Y is digit (SSD1 , SSD2)


      if(y == 3){

                  /* turn on SSD 1(LEFT).*/
                   /* turn on ODR*/
                   GPIOB->ODR |= (0x100);

                  /* turn off SSD 2.*/
                   /* turn off ODR*/
                   GPIOB->BRR |= (0x4);

                  /* turn off SSD 3.*/
                   /* turn off ODR*/
                   GPIOB->BRR |= (0x1);

                  /* turn off SSD 4.*/
                   /* turn off ODR*/
                   GPIOB->BRR |= (0x2);

                    SwitchSSD(x);
  }

      if(y == 2){

                  /* turn off SSD 1(LEFT).*/
                   /* turn off ODR*/
                   GPIOB->BRR |= (0x100);

                  /* turn on SSD 2.*/
                   /* turn on ODR*/
                   GPIOB->ODR |= (0x4);

                  /* turn off SSD 3.*/
                   /* turn off ODR*/
                   GPIOB->BRR |= (0x1);

                  /* turn off SSD 4.*/
                   /* turn off ODR*/
                   GPIOB->BRR |= (0x2);

                    SwitchSSD(x);
  }

      if(y == 1){

                  /* turn off SSD 1(LEFT).*/
                   /* turn off ODR*/
                   GPIOB->BRR |= (0x100);

                  /* turn off SSD 2.*/
                   /* turn off ODR*/
                   GPIOB->BRR |= (0x4);
                   /* turn on SSD 3.*/
                    /* turn on ODR*/
                   GPIOB->ODR |= (0x1);

                   /* turn off SSD 4.*/
                    /* turn off ODR*/
                   GPIOB->BRR |= (0x2);

                    SwitchSSD(x);
  }

      if(y == 0){

                   /* turn off SSD 1(LEFT).*/
                    /* turn off ODR*/
                   GPIOB->BRR |= (0x100);

                   /* turn off SSD 2.*/
                    /* turn off ODR*/
                   GPIOB->BRR |= (0x4);

                   /* turn off SSD 3.*/
                    /* turn off ODR*/
                   GPIOB->BRR |= (0x1);

                   /* turn on SSD 4.*/
                    /* turn on ODR*/
                   GPIOB->ODR |= (0x2);

                    SwitchSSD(x);
  }

}
void BSP_System_init() {

      __disable_irq();
      SystemCoreClockUpdate();
      BSP_led_init();
      SysTick_Config(SystemCoreClock / 1000);
    __enable_irq();
}

void init_adc() {
RCC->APBENR2 |= (1U << 20); //enable rcc for adc
RCC->IOPENR = (1U << 1); //enable GPIOB
//PB1 pin for adc in analog mode (by default)
ADC1->CR=0; //reset adc cr
ADC1->CFGR1 = 0;//reset adc cfgr1
ADC1 ->CR |= (1U << 28); // Enable adc voltage regulator
delay(500); //delay >20 us

//enable calibration, wait until completion
ADC1->CR |= (1U << 31); //calibration enable
while(((ADC1->CR>>31)==1));//Wait until calibration.
//enable end of cal. or sequence interrupts
// ADC1->IER |= (1U << 3); //end of conversion sequence interrupt
ADC1->IER |= (1U << 11); //end of calibration interrupt
// select resolution [conf. bit sample (6,8,10,12)]
ADC1 ->CFGR1 |= (2U << 3);// ; 8bit
//conf. single/continuous;
ADC1->CFGR1 &= ~(1U << 13);//cont=0;
ADC1->CFGR1 &= ~(1U << 16);//discen =8; single
//select sampling time from SMPR
ADC1->SMPR = (0 << 0);//SMP1
// ADC1->SMPR |= (10 << 4);//SMP2

//select tim trgo
ADC1->CFGR1 |= (3U << 6); //TGRO (extsel); 0xb011=3U for TIM3_TRGO
ADC1->CFGR1 |= (1U << 10); //Choose detect at rising edge (exten); 01
```

```
//enable channels (for the Anx pins)
ADC1->CFGR1 |= (9U << 26);//analog input channel 9; PB1
ADC1->CHSELR |= (1U << 9);//analog input channel 9; PB1
//Clear the ADRDY bit in ADC_ISR register by programming this bit to 1.
ADC1->ISR |= (1 << 0);
//enable adc and wait until it is ready
ADC1->CR |= (1 << 0);
while( (ADC1->ISR & (1 << 0)));
//Start conversion
ADC1->CR |= (1U << 2);
NVIC_SetPriority(ADC1_IRQn, 2); //Set priority to 2
NVIC_EnableIRQ(ADC1_IRQn); //Enable NVIC for TIM1
}

unsigned int ADC_start(void){

      ADC1->CR |= (1U << 2);                          /* Start ADC */
      while(!(ADC1->ISR & (1U << 2)));/* Is there any data? */
      return ADC1->DR;                                /* Data from pin */
}

void I2C1_IRQHandler(void) {

      // only enters when error
}

void init_I2C(void) {

      GPIOB->MODER &= ~(3U << 2*8);
      GPIOB->MODER |= (2 << 2*8);
      GPIOB->OTYPER |= (1U << 8);
      // choose AF from mux
      GPIOB->AFR[1] &= ~(0XFU<< 4*0);
      GPIOB->AFR[1] |= (6 << 4*0);

      // setup PB9 as AF6
      GPIOB->MODER &= ~(3U << 2*9);
      GPIOB->MODER |= (2 << 2*9);
      GPIOB->OTYPER |= (1U << 9);
      // choose AF6 from mux
      GPIOB->AFR[1] &= ~(0XFU<< 4*1);
      GPIOB->AFR[1] |= (6 << 4*1);

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

    I2C1->CR1 = 0;
    I2C1->CR1 |= (1U << 7); // ERRI

    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); // PF

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

void read_I2C(uint8_t devAddr, uint8_t regAddr, uint8_t *data, uint32_t num){
//WRITE OPERATION (Send address and register to read)
I2C1->CR2 = 0;
I2C1->CR2 |= ((uint32_t) devAddr << 1); // slave address
I2C1->CR2 |= (1U << 16); // Number of byte
I2C1->CR2 |= (1U << 13); // Generate Start
while(!(I2C1->ISR & (1 << 1))); // TXTS
```

```
I2C1 ->TXDR = (uint32_t) regAddr;
while(!(I2C1->ISR & (1 << 6))); // TC

// READ OPERATION (read data)
I2C1->CR2 = 0;
I2C1->CR2 |= ((uint32_t) devAddr << 1);
I2C1->CR2 |=(1U << 10); // READ mode
I2C1->CR2 |=(num << 16); // Number of bytes
I2C1->CR2 |=(1U << 15); // NACK
I2C1->CR2 |=(1U << 25); // AUTOEND
I2C1->CR2 |=(1U << 13); // Generate Start
for(size_t i=0; i<num; i++){
while(!(I2C1->ISR & (1 << 2))); // wait until RXNE =1

}
}


void write_general(uint8_t devAddr , uint16_t num , uint8_t* data){

      //WRITE OPERATION (Send address and register to read)
      I2C1->CR2 = 0;
      I2C1->CR2 |= ((uint32_t)devAddr << 1);//slave address
      I2C1->CR2 |= (3U << 16); // Number of byte
      I2C1->CR2 |= (1U << 25); // AUTOEND
      I2C1->CR2 |= (1U << 13); // Generate Start

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

void read_write_data(){

uint8_t data[10]; // stack , not zero , garbage data

read_I2C(LC512_ADDRESS , LC512_WHO_AM_I , data , 1);

read_I2C(LC512_ADDRESS , LC512_PWR_MGMT_1 , data , 1);

write_general(LC512_ADDRESS , LC512_PWR_MGMT_1, 0x00);
delay_ms(1000);

read_I2C(LC512_ADDRESS , LC512_PWR_MGMT_1 , data , 1);

}

/*
void enableEEPROM(uint16_t regAddr,uint8_t data){
      data[0]=I2C1->CR2 | ((uint32_t)devAddr << 1);//regADDRESS high
    data[1]=I2C1->CR2 | ((uint32_t)devAddr << 0);//regAddress low
      data[2]=(uint32_t)regAddr;//value of regADDRESS
    data[3]=(uint32_t)regAddr;//VALUE for regAddress

      //write to address 0x100
      data[0]=1;
      data[0]=0x00;
      data[1]=0;
      write_general(EEPROM_ADDRESS,data,3);
}
*/

void clearRowsKeypad(void){
           /* Clearing the rows here */
            GPIOB->ODR &= ~(1U << 9);   /// PB9
               GPIOB->ODR &= ~(1U << 5);   /// PB5
               GPIOB->ODR &= ~(1U << 4);   /// PB4
               GPIOB->ODR &= ~(1U << 3);   /// PB3
}
```

```
void setRowsKeypad(void){
         /* Setting the rows here   */
               GPIOB->ODR |= (1U << 9);   /// PB9
            GPIOB->ODR |= (1U << 5);   /// PB5
               GPIOB->ODR |= (1U << 4);   /// PB4
               GPIOB->ODR |= (1U << 3);   /// PB3
}
```