GEBZE TEKNİK ÜNİVERSİTESİ

ELEKTRONİK MÜHENDİSLİĞİ

ELEC-334

PROJE #01

Hazırlayanlar
161024093 - Emre Fidan
161024104 – Ertuğrul Garipardıç
1801022025 – Görkem Salih Ergün
1801022077 – Burak Kamil Çiftçi

Introduction

Within the scope of this project, keypad, ssd, M-M jumper cable and breadboard were used. A detailed flowchart and to-do list has been created. The project was completed by adding the items in this list step by step to the code. Within the scope of the project, the use of interrupt and timer was reinforced and learned. Separate libraries were created for SSD and Keypad components to ensure project code comprehension. The project was successfully concluded by ensuring the integration of the Keypad and SSD.

Problem

In this project, it is aimed to create a function generator.

The block diagram created for the project is as follows.

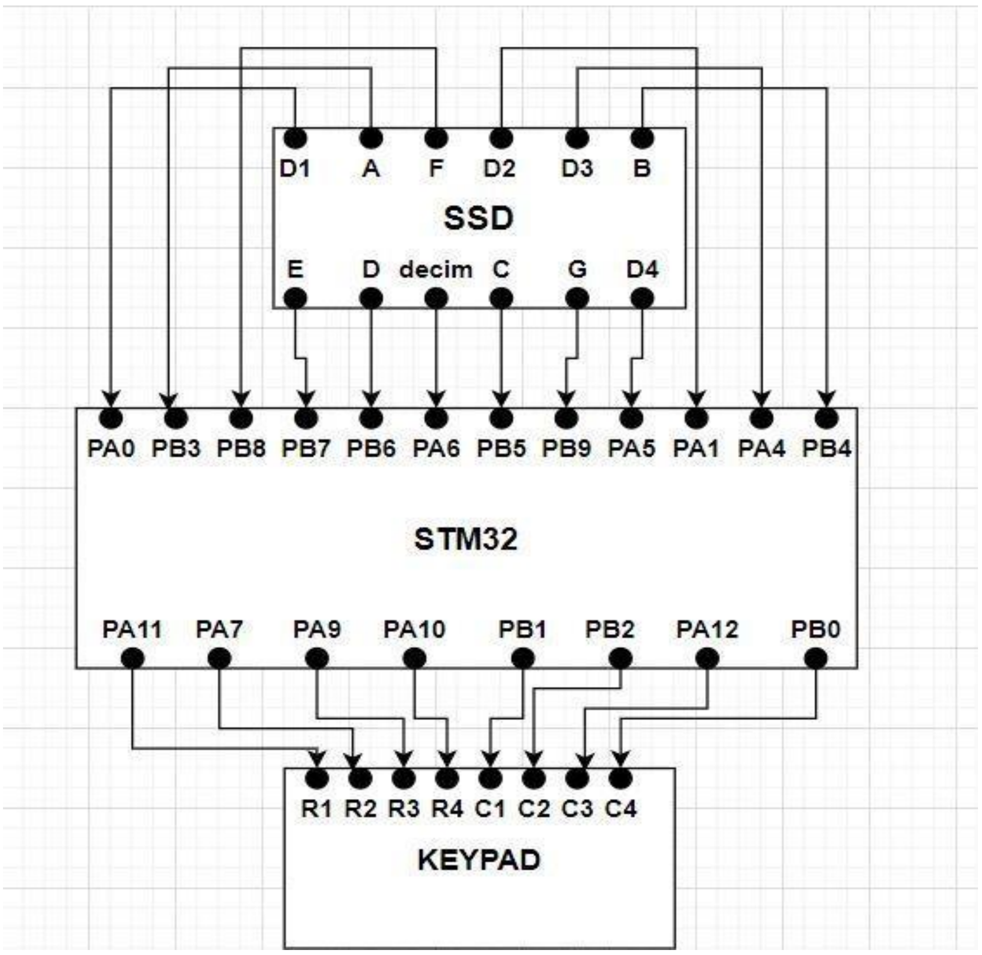


fig 1: Block Diagram for the Project

The schematic diagram created for the project is as follows.

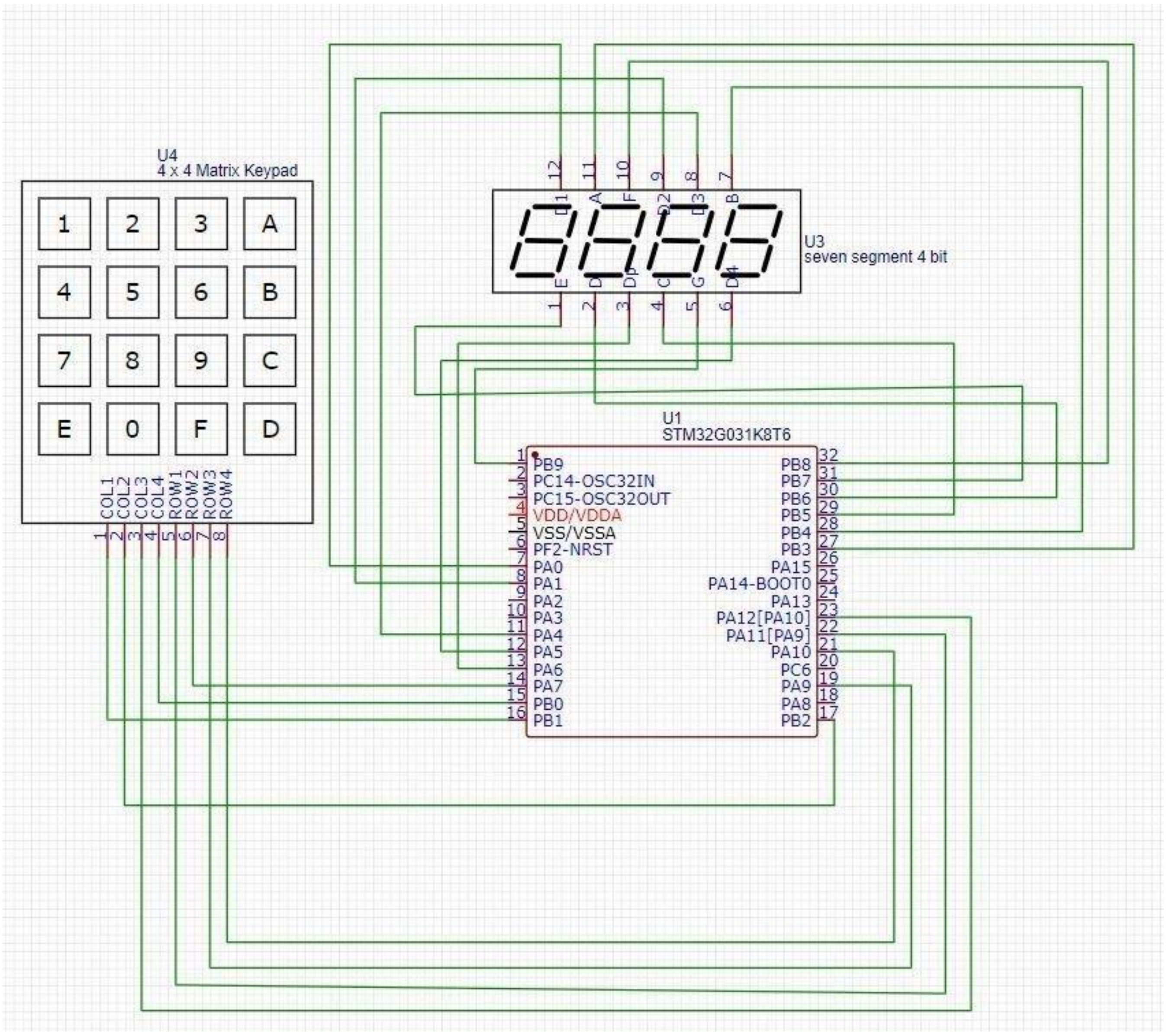


fig 2: Schematic Diagram for the Project

A to-do list and flowchart were created in the coding part for the problem. The project was written in C code on the STM32CubeIDE compiler.

The flowchart prepared for the project code is as follows.

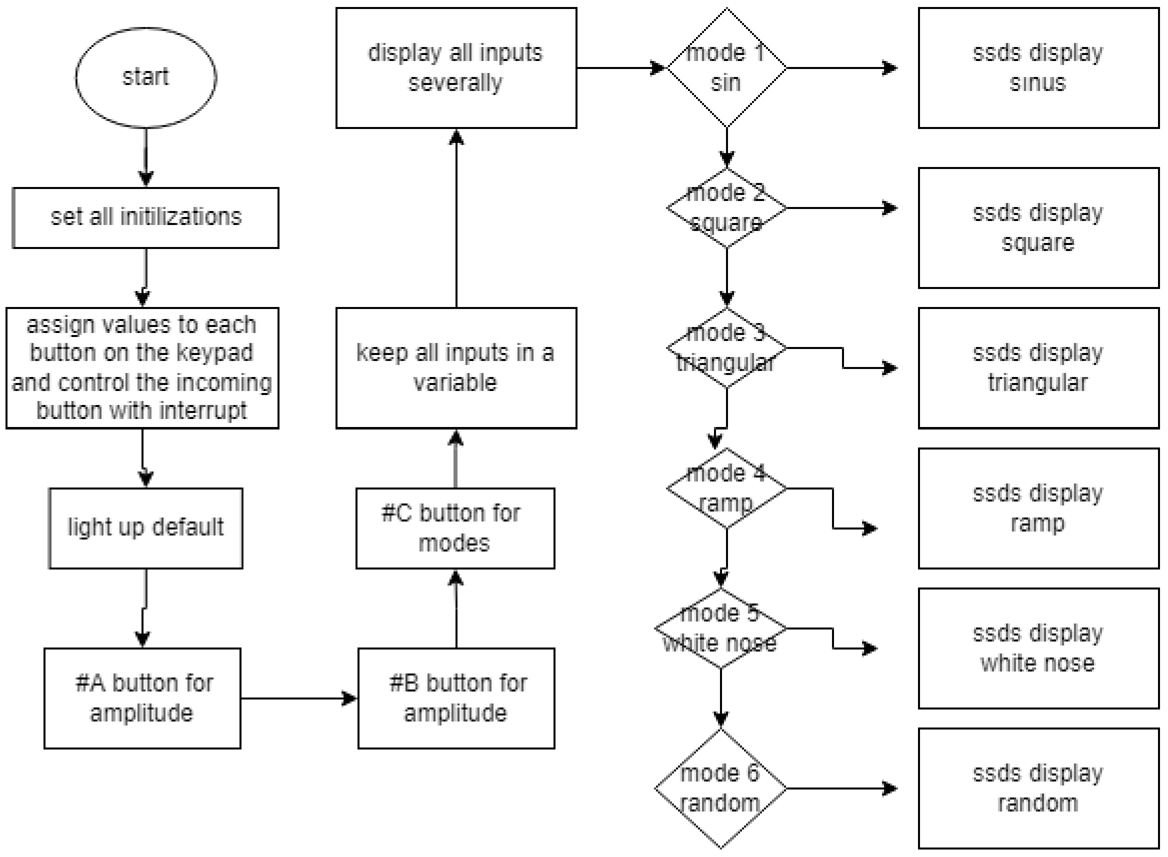


fig 3: Flowchart for the Project Code

STM32 microcontroller, SSD, Keypad, M-M jumper cable and breadboard are used in the circuit structure part of the project.

Keypad:

4x4 keypad consists of 8 terminals, 4 rows and 4 columns. These 8 PINS are driven out from 16 buttons present in the MODULE. Those 16 alphanumeric digits on the MODULE surface are the 16 buttons arranged in MATRIX formation. The working principle scheme is as follows.

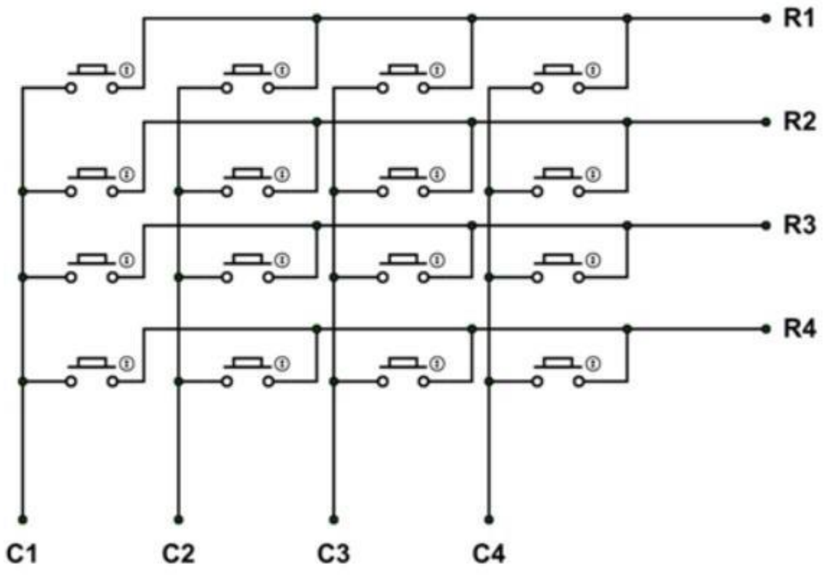


fig 4: Keypad Working Principle Scheme

4x4 Keypad schematic view is as follows.

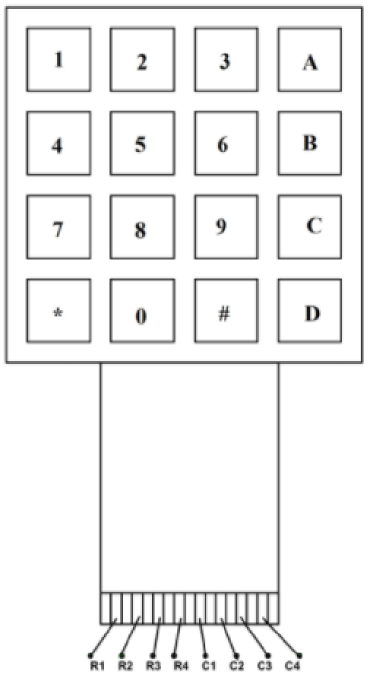
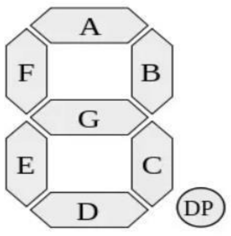


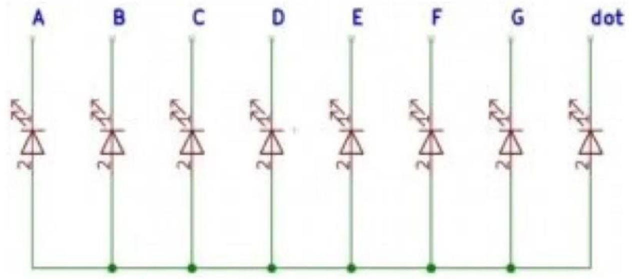
fig 5: Keypad Schematic View

Seven Segment Display:

Due to the SSD name, each digit contains 7 segments. These segments are indicated by letters A-G. A digit of SSD is as in the figure.



Each segment has its own output pin and the segment to be lit is set. In 4 digit SSD, each digit has its own output pin (D1-D2-D3-D4) Thus, the desired Digit to be lit can be activated. The visual describing the SSD working principle is as follows.



4 digit SSD schematic view is as follows.

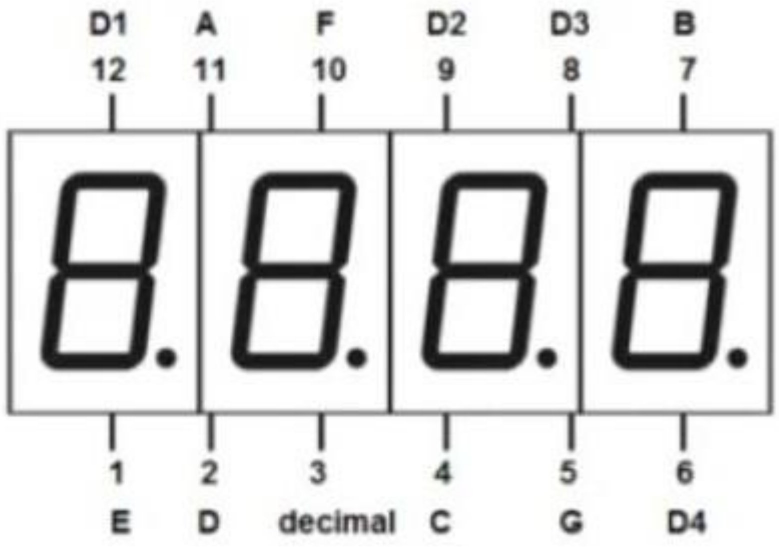
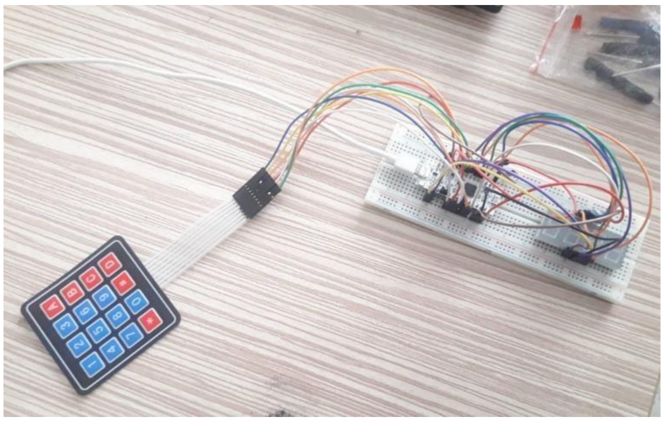


fig 6: 4-Digit Seven Segment Display Schematic View

Among the components described above, there are 12 output pins on the SSD. There are 8 pins in total, 4 outputs and 4 inputs on the keypad.

Pins are reserved for these components on STM. The connection table of these pins is as follows. Two photos of circuit setup are as follows.



Circuit Components and connections list table is as follows.

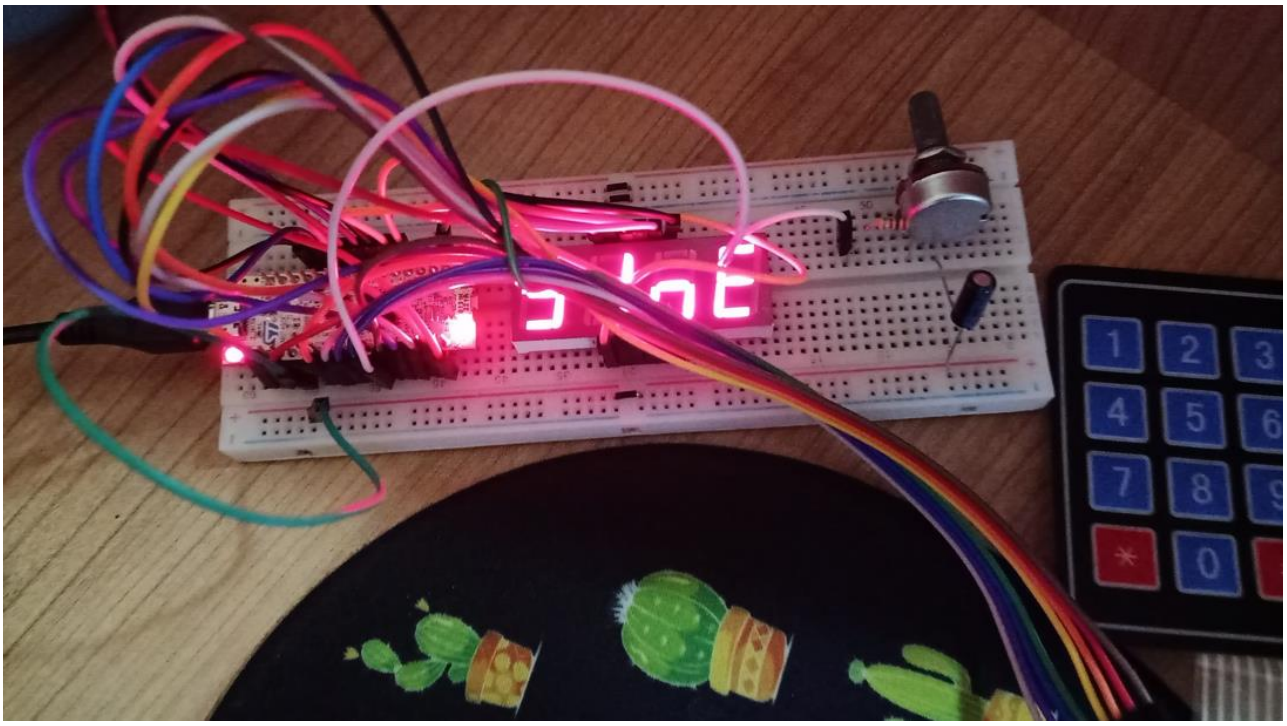
Seven Segment Display Pins	STM32 pins
A	PB3
B	PB4
C	PB5
D	PB6
E	PB7
F	PB8
G	PB9
D1	PA0
D2	PA1
D3	PA4
D4	PA5

Keypad Pins	STM32 pins
R1	PA11
R2	PA7
R3	PA9
R4	PA10
C1	PB1
C2	PB2
C3	PA12
C4	PB0

The to-do list within the scope of the project leaflet has been filled as follows.

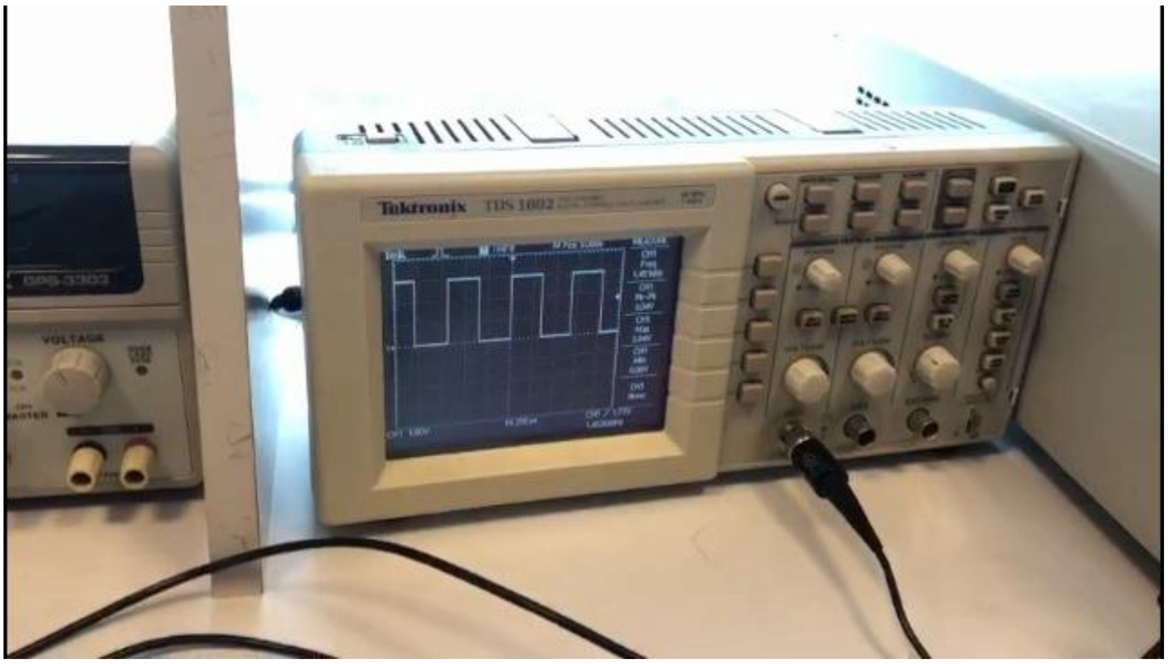
- ✓ Create function generator
- ✓ Produce sine, square, triangular, square, ramp, noise
- ✓ Implement multiple interrupts
- ✓ Generate analog signal using modulation
- ✓ Building and design RC-LPF filter
- ✓ Set # key for Enter key
- ✓ Set * Key for dot key
- ✓ Set A button for take Amplitude value
- ✓ Set B button for take Frequency value
- ✓ Set C button for cycle mode
- ✓ Set D button displaying mode
- ✓ Timeout for 10 seconds
- ✓ Digital stream output
- ✓ Operations could see on the SSDs
- ✓ SSD display mode for default
- ✓ No bouncing on the buttons
- ✓ Zero brightness difference on segments

CIRCUIT PHOTOS

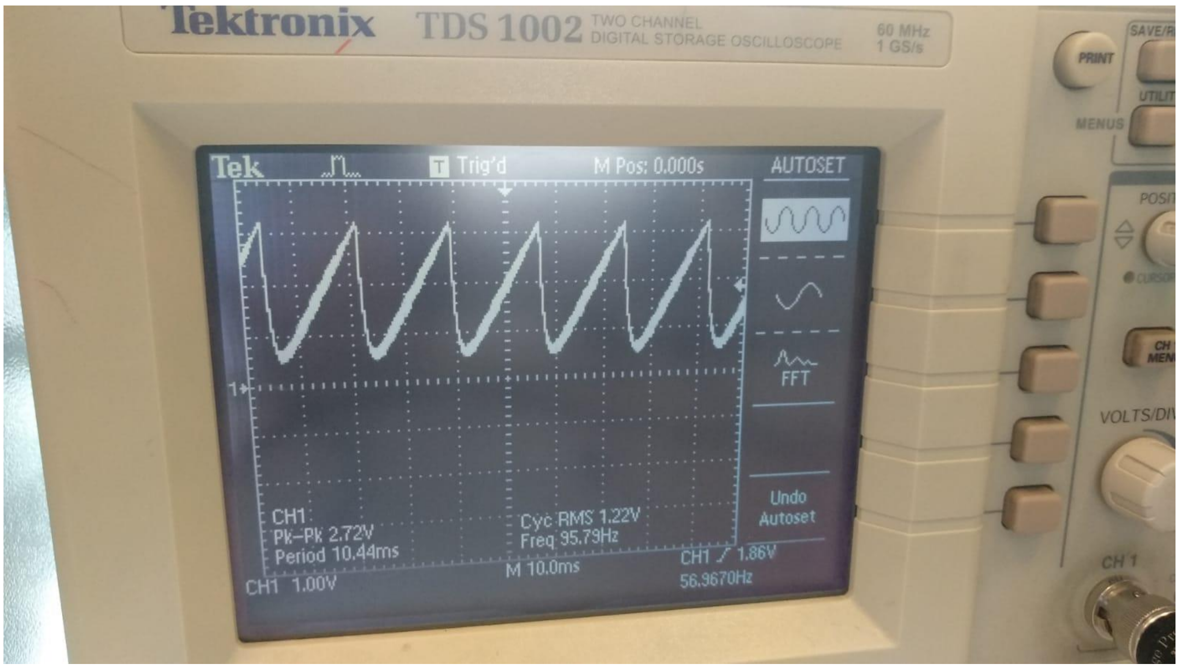


OSILLOSCOPE SCREENS

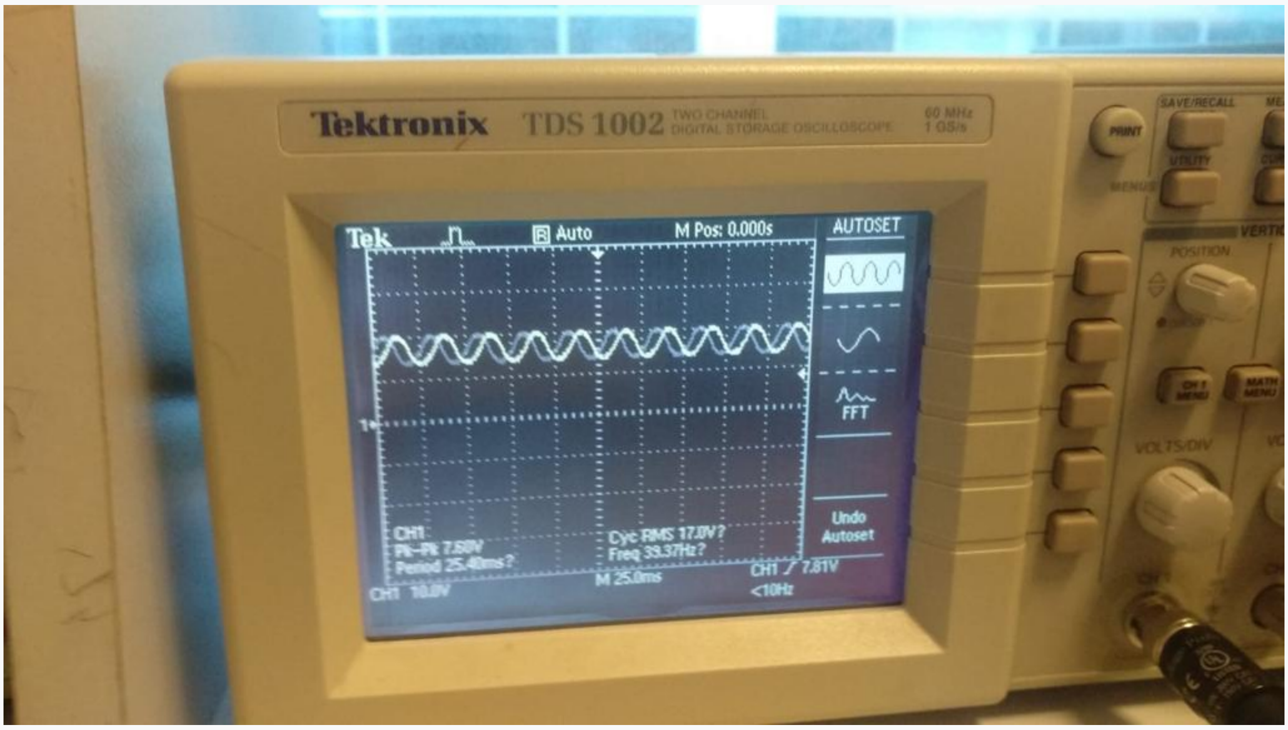
For Square Wave



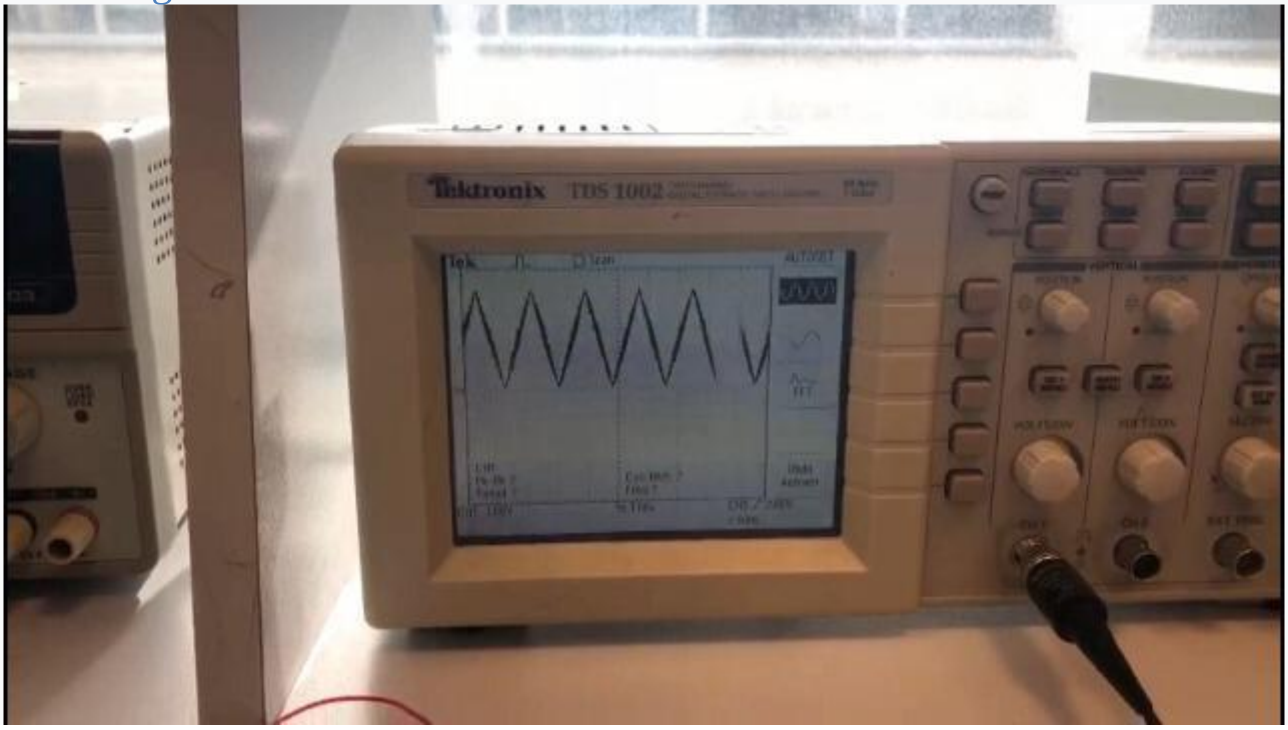
For Sawtooth Wave



For Sinusoidal wave



For triangle wave



CODE

Main.c

```
        #include "bsp.h"
        int main(void) {
                init_all();
                PWM_init();
                init_timer1();
           while(1){
                compare_display();
           }
           return 0;
}
```

bsp.h

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


        void init_all();
        void init_clocks();
        void init_ext_interrupts();
        void set_onboardLED_output();

        void turn_on_PC6(); // turns on on-boadr led
        void turn_off_PC6(); // turns off on-boadr led
        void toggle_PC6(); // toggles on-boadr led

        void set_output(int,int); //set the given pin to output mode (for A ports put 1,for B 2)
        void set_input(int,int); //set the given pin to input mode (for A ports put 1,for B 2)

        void set_pin_to1(int,int); //sets the given pin to logic 1 (for A ports put 1,for B 2)
        void set_pin_to0(int,int); //sets the given pin to logic 0 (for A ports put 1,for B 2)
        void toggle_pin(int,int);

        void USART2_IRQHandler(); //USART2 handler
        void printChar(uint8_t); // prints a character to console
        void _print( char*, int); //prints a given string with given length
        void print(char* buf); //prints given string

        void ResetKeypad(); //set all rows-outputs to zero
        void SetKeypad(); //set all rows-outputs to one
        void EXTI0_1_IRQHandler(); //external interrupt handler for PA0 and PB1
        void EXTI4_15_IRQHandler(); //external interrupt handler for PB7 and PB8

        void SetDigit1(); //activates first digit
```

```
void SetDigit2(); //activates second digit
void SetDigit3(); //activates third digit
void SetDigit4(); //activates fourth digit

void deSetDigit1();
void deSetDigit2();
void deSetDigit3();
void deSetDigit4();

void zero(); // displays zero
void one();// displays one
void two();// displays two
void three();// displays three
void four();// displays four
void five();// displays five
void six();// displays six
void seven();// displays seven
void eight();// displays eight
void nine();// displays nine
void black(); // clears SSD

void compare_display(); //
void Find_Digit_Places(int); // separates digits
void display_result(); // displays results
void overflow(); // displays OufL
void invalid(); // displays InVd
void control_negativity(); //checks is result is negative
void add();
void sub();
void mult();
void div();
void my_log();
void my_ln();
void my_sqrt();
void my_square();
void my_sin();
void my_cos();
void my_tan();
void my_cot();
void call_operators(); // calls necessary operator
void find_digit(int,int,int,int); // displays the digits on SSD
void display_digi();
void display_toth();
void display_sqre();
void display_gaus();
```

```
        void display_tria();
        void display_sine();
        void display_invalid(); //function that displays InVd for few at the screen when an overflow happens
        void display_mode(int mode);
        void TIM2_IRQHandler(void);
        void ModeSelect();

        void delay(int);

#endif
```

bsp.c

```
        #include "bsp.h"
        #include <math.h>

        volatile int Amplitude_A,Frequency_B,Mode_C,Display_D,enter,m,a,b;
        volatile int mode = 5;
        volatile int Amplitude[4],Frequency[4];
        volatile int i = 0;
        volatile int j = 0;
        volatile double entered_numbers[2] = {0.0};
        volatile double result;
        volatile int operator, equal, minus, sci, check_dot, presses = 0;
        volatile int operator2, dot_count, reset_counter, dot;
        int active_table[128];
        const uint32_t sin_table[128] = {
        512, 537, 562, 587, 612, 637, 661, 685, 709, 732, 754, 776, 798, 818, 838,
        857, 875, 893, 909, 925, 939, 952, 965, 976, 986, 995, 1002, 1009, 1014, 1018,
        1021, 1023, 1023, 1022, 1020, 1016, 1012, 1006, 999, 990, 981, 970, 959, 946, 932,
        917, 901, 884, 866, 848, 828, 808, 787, 765, 743, 720, 697, 673, 649, 624,
        600, 575, 549, 524, 499, 474, 448, 423, 399, 374, 350, 326, 303, 280, 258,
        236, 215, 195, 175, 157, 139, 122, 106, 91, 77, 64, 53, 42, 33, 24,
        17, 11, 7, 3, 1, 0, 0, 2, 5, 9, 14, 21, 28, 37, 47,
        58, 71, 84, 98, 114, 130, 148, 166, 185, 205, 225, 247, 269, 291, 314,
        338, 362, 386, 411, 436, 461, 486, 511
        };

        // All digits are deactive
        volatile int Digit1 = -1;
        volatile int Digit2 = -1;
        volatile int Digit3 = -1;
        volatile int Digit4 = -1;
        volatile int duty = 1;


        void init_clocks() {
```

```
  RCC -> IOPENR |= 7U;
}

void set_onboardLED_output() {
  GPIOC -> MODER &= ~(3U << 2 * 6);
  GPIOC -> MODER |= (1U<< 2 * 6);
}

void turn_on_PC6() {
  GPIOC -> ODR |= (1U<< 6);
}

void turn_off_PC6() {
  GPIOC -> ODR &= ~(1U<< 6);
}

void toggle_PC6() {
  GPIOC -> ODR ^= (1U<< 6);
}

void set_output(int port, int pin) {
switch (port) {
  case 1:
    GPIOA -> MODER &= ~(3U << 2 * pin);
    GPIOA -> MODER |= (1U<< 2 * pin);
    break;
  case 2:
    GPIOB -> MODER &= ~(3U << 2 * pin);
    GPIOB -> MODER |= (1U<< 2 * pin);
    break;
  }
}

void set_input(int port, int pin) {
  switch (port) {

  case 1:
    GPIOA -> MODER &= ~(3U << 2 * pin);
    break;
  case 2:
    GPIOB -> MODER &= ~(3U << 2 * pin);
    break;
  }
}
```

```
void set_pin_to1(int port, int pin) { //set the given output port to logic 1
  switch (port) {

  case 1:
    GPIOA -> ODR |= (1U<< pin);
    break;
  case 2:
    GPIOB -> ODR |= (1U<< pin);
    break;
  }
}

void set_pin_to0(int port, int pin) { //set the given output port to logic 0
  switch (port) {

  case 1:
    GPIOA -> ODR &= ~(1U<< pin);
    break;
  case 2:
    GPIOB -> ODR &= ~(1U<< pin);
    break;
  }
}

void toggle_pin(int port, int pin) {// toggles the value of given output port
  switch (port) {

  case 1:
    GPIOA -> ODR ^= (1U<< pin);
    break;
  case 2:
    GPIOB -> ODR ^= (1U<< pin);
    break;
  }
}

void delay(int time) {
  for (; time > 0; time--);
}

void my_delay() {
  for (int i = 0; i < 200; i++);
}

void ResetKeypad() {
  GPIOA -> ODR &= ~(1U<< 10);
  GPIOA -> ODR &= ~(1U<< 9);
  GPIOB -> ODR &= ~(1U<< 0);
  GPIOB -> ODR &= ~(1U<< 3);
}

void SetKeypad() {
  GPIOA -> ODR |= (1U<< 10);
  GPIOA -> ODR |= (1U<< 9);
  GPIOB -> ODR |= (1U<< 0);
  GPIOB -> ODR |= (1U<< 3);
}

void EXTI0_1_IRQHandler() {
  //PA10 PA9 PB0 PB8 outputs - rows
  delay(200);
  reset_counter = 0;


  if ((EXTI -> RPR1 >> 0) & 1) {
    ResetKeypad();

    // Row 1 - number 1
    toggle_pin(1,10);
    if ((GPIOA -> IDR >> 0) & 1) {
        if(Amplitude_A==1){
            Amplitude[i]=1;
            i++;
        }
        if(Frequency_B==1){
            Frequency[j]=1;
            j++;
        }
        if(Mode_C==1){
            mode=1;
        }
    }
    toggle_pin(1,10);

    // Row 2 - number 4
    toggle_pin(1, 9);
    if ((GPIOA -> IDR >> 0) & 1) {
        if(Amplitude_A==1){
            Amplitude[i]=4;
                i++;
```

```
                    }
                    if(Frequency_B==1){
                            Frequency[j]=4;
                            j++;
                    }
                    if(Mode_C==1){
                            mode=4;
                    }

  }
  toggle_pin(1, 9);

  // Row 3 - number 7
  toggle_pin(2, 0);
  if ((GPIOA -> IDR >> 0) & 1) {
        if(Amplitude_A==1){
                    Amplitude[i]=7;
                            i++;
                    }
                    if(Frequency_B==1){
                            Frequency[j]=7;
                            j++;
                    }
                    if(Mode_C==1){
                            mode=0;
                    }

  }
  toggle_pin(2, 0);

}

SetKeypad();

//check rows for Column 2
if ((EXTI -> RPR1 >> 1) & 1) { //check status register for PB1
  ResetKeypad();
  // Row 1 - number 2
  GPIOA -> ODR ^= (1U<< 10);
  if ((GPIOB -> IDR >> 1) & 1) {
        if(Amplitude_A==1){
                    Amplitude[i]=2;
                            i++;
                    }
                    if(Frequency_B==1){
```

```
                Frequency[j]=2;
                j++;
            }
            if(Mode_C==1){
                mode=2;
            }
}
GPIOA -> ODR ^= (1U<< 10);

// Row 2 - number 5
GPIOA -> ODR ^= (1U<< 9);
if ((GPIOB -> IDR >> 1) & 1) {
    if(Amplitude_A==1){
            Amplitude[i]=5;
                i++;
            }
            if(Frequency_B==1){
                Frequency[j]=5;
                j++;
            }
            if(Mode_C==1){
                mode=5;
            }
}
GPIOA -> ODR ^= (1U<< 9);

// Row 3 - number 8
toggle_pin(2, 0);
if ((GPIOB -> IDR >> 1) & 1) {
    if(Amplitude_A==1){
            Amplitude[i]=8;
                i++;
            }
            if(Frequency_B==1){
                Frequency[j]=8;
                j++;
            }
            if(Mode_C==1){
                mode=0;
            }
}
toggle_pin(2, 0);

// Row 4 - number 0
toggle_pin(2, 3);
```

```
  if ((GPIOB -> IDR >> 1) & 1) {
        if(Amplitude_A==1){
                  Amplitude[i]=0;
                        i++;
                  }
                  if(Frequency_B==1){
                        Frequency[j]=0;
                        j++;
                  }
                  if(Mode_C==1){
                        mode=0;
                  }
  }
  toggle_pin(2, 3);

}

SetKeypad();

if(Amplitude_A == 1 && Frequency_B != 1){
if(i==4){
                    Digit1=Amplitude[3];
                    Digit2=Amplitude[2];
                    Digit3=Amplitude[1];
                    Digit4=Amplitude[0];
            }
            if(i==3){
                    Digit1=-1;
                    Digit2=Amplitude[2];
                    Digit3=Amplitude[1];
                    Digit4=Amplitude[0];
                    }
            if(i==2){
                    Digit1=-1;
                    Digit2=-1;
                    Digit3=Amplitude[1];
                    Digit4=Amplitude[0];
                   }
            if(i==1){
                    Digit1=-1;
                    Digit2=-1;
                    Digit3=-1;
                    Digit4=Amplitude[0];
                  }
}
```

```
  if(Amplitude_A != 1 && Frequency_B == 1){
  if(j==4){
                  Digit1=Frequency[3];
                  Digit2=Frequency[2];
                  Digit3=Frequency[1];
                  Digit4=Frequency[0];
          }
          if(j==3){
                  Digit1=-1;
                  Digit2=Frequency[2];
                  Digit3=Frequency[1];
                  Digit4=Frequency[0];
                  }
          if(j==2){
                  Digit1=-1;
                  Digit2=-1;
                  Digit3=Frequency[1];
                  Digit4=Frequency[0];
                 }
          if(j==1){
                  Digit1=-1;
                  Digit2=-1;
                  Digit3=-1;
                  Digit4=Frequency[0];
                          }
  }

  //reset status registers
  EXTI -> RPR1 |= (1U<< 0);
  EXTI -> RPR1 |= (1U<< 1);

  //display the current number on the SSD
if (Amplitude_A == 1 || Frequency_B == 1){
        while(!( (EXTI ->  RPR1 >> 1) & 1 || (EXTI -> RPR1 >> 0) & 1 || ( (EXTI ->  RPR1 >> 7) & 1)))
        find_digit(Digit4,Digit3,Digit2,Digit1);
}



}

void EXTI4_15_IRQHandler() { //interrupt from PB7(Column 3) and PB8(Column 4)
  //PA10 PA9 PB0 PB8 outputs - rows
  delay(300);// for preventing bouncing
  reset_counter = 0; // reset 10sec counter if an interrupt happens
```

```
//check rows for column 3
if ((EXTI -> RPR1 >> 7) & 1) {
  ResetKeypad();
  // Row 1 - number 3
  toggle_pin(1,10);
  if ((GPIOB -> IDR >> 7) & 1) {
        if(Amplitude_A==1){
                Amplitude[i]=3;
                        i++;
                }
                if(Frequency_B==1){
                        Frequency[j]=3;
                        j++;
                }
                if(Mode_C==1){
                        mode=3;
                }
  }
  toggle_pin(1,10);
  // Row 2 - number 6
  toggle_pin(1,9);
  if ((GPIOB -> IDR >> 7) & 1) {
        if(Amplitude_A==1){
                Amplitude[i]=6;
                        i++;
                }
                if(Frequency_B==1){
                        Frequency[j]=6;
                        j++;
                }
                if(Mode_C==1){
                        mode=6;
                }
  }
  toggle_pin(1,9);
  // Row 3 - number 9
  toggle_pin(2, 0);
  if ((GPIOB -> IDR >> 7) & 1) {
        if(Amplitude_A==1){
                Amplitude[i]=9;
                        i++;
                }
                if(Frequency_B==1){
                        Frequency[j]=9;
```

```
                  j++;
            }
            if(Mode_C==1){
                  mode=0;
            }
  }
  toggle_pin(2, 0);
  // Row 4 - # or E key
  toggle_pin(2, 3);
  if ((GPIOB -> IDR >> 7) & 1) {
      enter = 1;
      Amplitude_A=0;
      Frequency_B=0;
      Mode_C=0;
      Display_D=0;

    GPIOC -> ODR ^= (1U<< 6);

    toggle_pin(2, 3);
  }

}
SetKeypad();
//check rows for column 4
if ((EXTI -> RPR1 >> 8) & 1) {
  ResetKeypad();
  // A key
  GPIOA -> ODR ^= (1U<< 10);
  if ((GPIOB -> IDR >> 8) & 1) {
    Amplitude_A=1;
  }
  GPIOA -> ODR ^= (1U<< 10);
  // B key
  GPIOA -> ODR ^= (1U<< 9);
  if ((GPIOB -> IDR >> 8) & 1) {
    Frequency_B=1;
    Amplitude_A = 0;
    Digit1 = -1;
    Digit2 = -1;
    Digit3 = -1;
    Digit4 = -1;
  }
  GPIOA -> ODR ^= (1U<< 9);
  // C key
  toggle_pin(2, 0);
```

```
 if ((GPIOB -> IDR >> 8) & 1) {
   Mode_C=1;
   Amplitude_A = 0;
   Frequency_B = 0;
   Digit1 = -1;
   Digit2 = -1;
   Digit3 = -1;
   Digit4 = -1;
 }
 toggle_pin(2, 0);
 // D key
 toggle_pin(2, 3);
 if ((GPIOB -> IDR >> 8) & 1) {
     Mode_C=0;
     Amplitude_A = 0;
     Frequency_B = 0;
     Display_D=1;
 }
 toggle_pin(2, 3);
}

SetKeypad();



 if (Display_D==1){

           if(i==4){
                     Digit1=Amplitude[3];
                     Digit2=Amplitude[2];
                     Digit3=Amplitude[1];
                     Digit4=Amplitude[0];
           }
           if(i==3){
                     Digit1=Amplitude[3];
                     Digit2=Amplitude[2];
                     Digit3=Amplitude[1];
                     Digit4=Amplitude[0];
                     }
           if(i==2){
                     Digit1=Amplitude[3];
                     Digit2=Amplitude[2];
                     Digit3=Amplitude[1];
                     Digit4=Amplitude[0];
                    }
```

```
if(i==1){
        Digit1=Amplitude[3];
        Digit2=Amplitude[2];
        Digit3=Amplitude[1];
        Digit4=Amplitude[0];
               }
for(a=0;a<1600;a++){
find_digit(Digit4, Digit3, Digit2, Digit1);}

if(i==4){
                Digit1=Frequency[3];
                Digit2=Frequency[2];
                Digit3=Frequency[1];
                Digit4=Frequency[0];
        }
        if(i==3){
                Digit1=Frequency[3];
                Digit2=Frequency[2];
                Digit3=Frequency[1];
                Digit4=Frequency[0];
                }
        if(i==2){
                Digit1=Frequency[3];
                Digit2=Frequency[2];
                Digit3=Frequency[1];
                Digit4=Frequency[0];
               }
        if(i==1){
                Digit1=Frequency[3];
                Digit2=Frequency[2];
                Digit3=Frequency[1];
                Digit4=Frequency[0];



                        }

        for(a=0;a<1600;a++){
        find_digit(Digit4, Digit3, Digit2, Digit1);}

        for(a=0;a<1600;a++){
                // Mode
                display_mode(mode);
                i = 0; j = 0;
                Amplitude_A = 0;
                Frequency_B = 0;
```

```
                                Mode_C = 0;
                                Display_D = 0;
                                Digit1 = -1;
                                Digit2 = -1;
                                Digit3 = -1;
                                Digit4 = -1;
                                for(b=0;b<4;b++){Amplitude[b]=0; Frequency[b]=0;}
                        }
  }

if(Amplitude_A == 1 && Frequency_B != 1){
  if(i==4){
                        Digit1=Amplitude[3];
                        Digit2=Amplitude[2];
                        Digit3=Amplitude[1];
                        Digit4=Amplitude[0];
                }
                if(i==3){
                        Digit1=-1;
                        Digit2=Amplitude[2];
                        Digit3=Amplitude[1];
                        Digit4=Amplitude[0];
                        }
                if(i==2){
                        Digit1=-1;
                        Digit2=-1;
                        Digit3=Amplitude[1];
                        Digit4=Amplitude[0];
                        }
                if(i==1){
                        Digit1=-1;
                        Digit2=-1;
                        Digit3=-1;
                        Digit4=Amplitude[0];
                       }
}


if(Amplitude_A != 1 && Frequency_B == 1){
if(j==4){
                        Digit1=Frequency[3];
                        Digit2=Frequency[2];
                        Digit3=Frequency[1];
                        Digit4=Frequency[0];
                }
```

```
            if(j==3){
                    Digit1=-1;
                    Digit2=Frequency[2];
                    Digit3=Frequency[1];
                    Digit4=Frequency[0];
                    }
            if(j==2){
                    Digit1=-1;
                    Digit2=-1;
                    Digit3=Frequency[1];
                    Digit4=Frequency[0];
                   }
            if(j==1){
                    Digit1=-1;
                    Digit2=-1;
                    Digit3=-1;
                    Digit4=Frequency[0];
                  }
}

if(Mode_C == 1){
        while(!( (EXTI ->  RPR1 >> 1) & 1 || (EXTI -> RPR1 >> 0) & 1 || ( (EXTI ->  RPR1 >> 7) &
1))){

                if(m != 0){
                display_mode(mode);
        }

}
        m++;
}

  //reset status registers
  EXTI -> RPR1 |= (1U<< 0);
  EXTI -> RPR1 |= (1U<< 1);



 if (Amplitude_A == 1 || Frequency_B == 1){
                while(!( (EXTI ->  RPR1 >> 1) & 1 || (EXTI -> RPR1 >> 0) & 1 || ( (EXTI ->  RPR1 >>
7) & 1)))
        find_digit(Digit4,Digit3,Digit2,Digit1);
 }

  // Reset status registers
  EXTI -> RPR1 |= (1U<< 7);
  EXTI -> RPR1 |= (1U<< 8);
}

void init_all() {

  init_clocks();
  set_onboardLED_output();
  GPIOB -> MODER &= ~(3U << 2 * 2);
  // input keypad
  uint32_t x = 0x82AC0;
  GPIOB -> MODER &= ~x;
  // output keypad
  x = 0x41500;
  GPIOB -> MODER |= x;
  // input SSD
  x = 0x28AAA08;
  GPIOA -> MODER &= ~x;
  //output SSD
  x = 0x1455504;
  GPIOA -> MODER |= x;

  //PA10 PA9 PB0 PB3 outputs - rows
  set_output(1,10);
  set_output(1,9);
  set_output(2,0);
  set_output(2,3);

  //PA0 PB1 PB8 PB7 inputs - columns, enable pull-down
  set_input(1,0);
  GPIOA->PUPDR |= (2U << 2*0);
  set_input(2,1);
  GPIOB->PUPDR |= (2U << 2*1);
  set_input(2,7);
  GPIOB->PUPDR |= (2U << 2*7);
  set_input(2,8);
  GPIOB->PUPDR |= (2U << 2*8);

  init_ext_interrupts();

  //set input pins to 1 - columns
  set_pin_to1(1,0);
  set_pin_to1(2,1);
  set_pin_to1(2,7);
  set_pin_to1(2,8);
```

```
}

void init_ext_interrupts(){
  //PA0
  EXTI->EXTICR[0] |= (0U << 0);
  EXTI->RTSR1 |= (1U << 0);
  EXTI->IMR1 |= (1U << 0);
  NVIC_SetPriority(EXTI0_1_IRQn, 0);
  NVIC_EnableIRQ(EXTI0_1_IRQn);

  //PB1
  EXTI->EXTICR[0] |= (1U << 8*1);
  EXTI->RTSR1 |= (1U << 1);
  EXTI->IMR1 |= (1U << 1);
  NVIC_SetPriority(EXTI0_1_IRQn, 0);
  NVIC_EnableIRQ(EXTI0_1_IRQn);

          //PB7
  EXTI->EXTICR[1] |= (1U << 8*3);
  EXTI->RTSR1 |= (1U << 7);
  EXTI->IMR1 |= (1U << 7);
  NVIC_SetPriority(EXTI4_15_IRQn, 0);
  NVIC_EnableIRQ(EXTI4_15_IRQn);

          //PB8
  EXTI->EXTICR[2] |= (1U << 8*0);
  EXTI->RTSR1 |= (1U << 8);
  EXTI->IMR1 |= (1U << 8);
  NVIC_SetPriority(EXTI4_15_IRQn, 0);
  NVIC_EnableIRQ(EXTI4_15_IRQn);
}

void SetDigit1() {
  GPIOB -> ODR |= (1U<< 4);
}
void SetDigit2() {
  GPIOB -> ODR |= (1U<< 5);
}
void SetDigit3() {
  GPIOB -> ODR |= (1U<< 9);
}
void SetDigit4() {
  GPIOB -> ODR |= (1U<< 6);
}
void deSetDigit1() {
  GPIOB -> ODR &= ~(1U<< 4);
}
void deSetDigit2() {
  GPIOB -> ODR &= ~(1U<< 5);
}
void deSetDigit3() {
  GPIOB -> ODR &= ~(1U<< 9);
}
void deSetDigit4() {
  GPIOB -> ODR &= ~(1U<< 6);
}


void zero() {
  uint32_t x = 0x11D2;
  GPIOA -> ODR &= ~(x);
}

void one() {
	// B=C=0, A=D=E=F=G=1
	uint32_t x = 0x1080;
	GPIOA -> ODR &= ~(x);
}

void two() {
	// 1001 1001 0010
	uint32_t x = 0x992;
	GPIOA -> ODR &= ~(x);
}

void three() {
	// 1 1001 1001 0000
	uint32_t x = 0x1990;
	GPIOA -> ODR &= ~(x);
}

void four() {
	// 1 1000 1100 0000
	uint32_t x = 0x18C0;
	GPIOA -> ODR &= ~(x);
}

void five() {
	//1 1001 0101 0000
	uint32_t x = 0x1950;
	GPIOA -> ODR &= ~(x);
}
void six() {
	//1 1001 0101 0010
	uint32_t x = 0x1952;
	GPIOA -> ODR &= ~(x);
}

void seven() {
	//1 0001 1000 0000
	uint32_t x = 0x1180;
	GPIOA -> ODR &= ~(x);
}

void eight() {
	//1 1001 1101 0010
	uint32_t x = 0x19D2;
	GPIOA -> ODR &= ~(x);
}

void nine() {
	//1 1001 1101 0000
	uint32_t x = 0x19D0;
	GPIOA -> ODR &= ~(x);
}

//deactivate all digits and segments
void blackout() {
	//10 0111 0000
	uint32_t x = 0x270;
	GPIOB -> ODR &= ~(x);
	x = 0x19F2;
	GPIOA -> ODR |= x;
}




void compare_display() {

  while (1) {
    while (Amplitude_A != 1)
	display_mode(mode);
  }
}
```

```
void find_digit(int Digit1, int Digit2, int Digit3, int Digit4) {
  switch (Digit1) {
  case 0:
    blackout();
    SetDigit1();
    zero();
    my_delay();
    break;
  case 1:
    blackout();
    SetDigit1();
    one();
    my_delay();
    break;
  case 2:
    blackout();
    SetDigit1();
    two();
    my_delay();
    break;
  case 3:
    blackout();
    SetDigit1();
    three();
    my_delay();
    break;
  case 4:
    blackout();
    SetDigit1();
    four();
    my_delay();
    break;
  case 5:
    blackout();
    SetDigit1();
    five();
    my_delay();
    break;
  case 6:
    blackout();
    SetDigit1();
    six();
    my_delay();
    break;
```

```
case 7:
  blackout();
  SetDigit1();
  seven();
  my_delay();
  break;
case 8:
  blackout();
  SetDigit1();
  eight();
  my_delay();
  break;
case 9:
  blackout();
  SetDigit1();
  nine();
  my_delay();
  break;
case 10:
  blackout();
  SetDigit1();
  GPIOA -> ODR &= ~(1U<< 11);
  my_delay();
  break;
default:
  deSetDigit1();
}

switch (Digit2) {
case 0:
  blackout();
  SetDigit2();
  zero();
  my_delay();
  break;
case 1:
  blackout();
  SetDigit2();
  one();
  my_delay();
  break;
case 2:
  blackout();
  SetDigit2();
  two();
  my_delay();
  break;
case 3:
  blackout();
  SetDigit2();
  three();
  my_delay();
  break;
case 4:
  blackout();
  SetDigit2();
  four();
  my_delay();
  break;
case 5:
  blackout();
  SetDigit2();
  five();
  my_delay();
  break;
case 6:
  blackout();
  SetDigit2();
  six();
  my_delay();
  break;
case 7:
  blackout();
  SetDigit2();
  seven();
  my_delay();
  break;
case 8:
  blackout();
  SetDigit2();
  eight();
  my_delay();
  break;
case 9:
  blackout();
  SetDigit2();
  nine();
  my_delay();
  break;
default:
  deSetDigit2();
}

switch (Digit3) {
case 0:
  blackout();
  SetDigit3();
  zero();
  my_delay();
  break;
case 1:
  blackout();
  SetDigit3();
  one();
  my_delay();
  break;
case 2:
  blackout();
  SetDigit3();
  two();
  my_delay();
  break;
case 3:
  blackout();
  SetDigit3();
  three();
  my_delay();
  break;
case 4:
  blackout();
  SetDigit3();
  four();
  my_delay();
  break;
case 5:
  blackout();
  SetDigit3();
  five();
  my_delay();
  break;
case 6:
  blackout();
  SetDigit3();
  six();
  my_delay();
  break;
case 7:
  blackout();
  SetDigit3();
  seven();
  my_delay();
  break;
case 8:
  blackout();
  SetDigit3();
  eight();
  my_delay();
  break;
case 9:
  blackout();
  SetDigit3();
  nine();
  my_delay();
  break;
default:
  deSetDigit3();
}

switch (Digit4) {
case 0:
  blackout();
  SetDigit4();
  zero();
  my_delay();

  break;
case 1:
  blackout();
  SetDigit4();
  one();
  my_delay();
  break;

case 2:
  blackout();
  SetDigit4();
  two();
  my_delay();

  break;
```

```
case 3:
  blackout();
  SetDigit4();
  three();
  my_delay();

  break;
case 4:
  blackout();
  SetDigit4();
  four();
  my_delay();

  break;
case 5:
  blackout();
  SetDigit4();
  five();
  my_delay();

  break;
case 6:
  blackout();
  SetDigit4();
  six();
  my_delay();

  break;
case 7:
  blackout();
  SetDigit4();
  seven();
  my_delay();

  break;
case 8:
  blackout();
  SetDigit4();
  eight();
  my_delay();

  break;
case 9:
  blackout();
  SetDigit4();
```

```
    nine();
    my_delay();

    break;
  default:
    deSetDigit4();
  }
}

void display_mode(int mode){

            if(mode==1){

                        display_sine();

            }
            else if(mode==2){
                        display_sqre();

            }
            else if(mode==3){
                        display_tria();
            }
            else if(mode==4){
                        display_toth();

            }
            else if(mode==5){
                        display_gaus();
            }
            else if(mode==6){
                        display_digi();
            }
            else{
                        display_invalid();

}}


void display_invalid() {
  volatile uint32_t x;
    blackout();
    SetDigit1();
    zero(); // I
    my_delay();
    blackout();
    SetDigit2();
    zero();
    my_delay();
    blackout();
    SetDigit3();
    zero();
    my_delay();
    blackout();
    SetDigit4();
    zero();
    my_delay();
    blackout();
}

void display_sine(){
            volatile uint32_t x;



            blackout();
            SetDigit1();
            five(); // 5
            my_delay();
            blackout();
            SetDigit2();
            one(); // I
            my_delay();
            blackout();
            SetDigit3();
            x = 0x1802; // 0001 1000 0000 0010 N 1 11 12
   GPIOA -> ODR &= ~(x);
   my_delay();
   blackout();
   SetDigit4();
   x = 0x952; //0000 0000 0101 0010 d
   GPIOA -> ODR &= ~(x);
   my_delay();
   blackout();
}
void display_tria(){
            volatile uint32_t x;



            blackout();
            SetDigit1();
            x = 0x1902; //T
            GPIOA -> ODR &= ~(x);
            my_delay();
            blackout();
            SetDigit2();
            x = 0x802; // R
               GPIOA -> ODR &= ~(x);
            my_delay();
blackout();
SetDigit3();
one();
my_delay();
blackout();
SetDigit4();
x = 0x19C2; //A
GPIOA -> ODR &= ~(x);
my_delay();
blackout();
}
void display_sqre(){
            volatile uint32_t x;
            blackout();
            SetDigit1();
            five();
            my_delay();
            blackout();
            SetDigit2();
            x = 0x19C0; // Q
            GPIOA -> ODR &= ~(x);
            my_delay();
            blackout();
            SetDigit3();
            x = 0x802; // R
            GPIOA -> ODR &= ~(x);
            my_delay();
            blackout();
            SetDigit4();
            x = 0x952; //E
            GPIOA -> ODR &= ~(x);
            my_delay();
            blackout();
}
void display_toth(){
            volatile uint32_t x;
            blackout();
```

```
		SetDigit1();
		x = 0x1902; //T
		GPIOA -> ODR &= ~(x);
		my_delay();
		blackout();
		SetDigit2();
		zero();
		my_delay();
		blackout();
		SetDigit3();
		x = 0x1902; //T
		GPIOA -> ODR &= ~(x);
		my_delay();
		blackout();
		SetDigit4();
		x = 0x18C2; //h
		GPIOA -> ODR &= ~(x);
		my_delay();
		blackout();
	}

	void display_digi(){
		volatile uint32_t x;
		blackout();
		SetDigit1();
	    x = 0x1892; //0000 0000 0101 0010 d
	    GPIOA -> ODR &= ~(x);
		my_delay();
		blackout();
		SetDigit2();
		one();
		my_delay();
		blackout();
		SetDigit3();
		six();
		my_delay();
		blackout();
		SetDigit4();
		one();
		my_delay();
		blackout();
	}
	void display_gaus(){
		volatile uint32_t x;
		blackout();
```

```
        SetDigit1();
        six();
        my_delay();
        blackout();
        SetDigit2();
        x = 0x19C2; //A
        GPIOA -> ODR &= ~(x);
        my_delay();
        blackout();
        SetDigit3();
        x = 0x10D2;
        GPIOA -> ODR &= ~(x);
        my_delay();
        blackout();
        SetDigit4();
        five();
        my_delay();
        blackout();
}

void PWM_init(void) {
 RCC -> IOPENR |= (2U << 0); // B open
 RCC -> PLLCFGR |= (2U << 0); //hsÄ±16
 RCC -> PLLCFGR |= (1U << 24); // set q n r constant
 RCC -> PLLCFGR |= (1U << 25);// SETQ
 RCC -> PLLCFGR |= (1U << 28);//ENABLE N
 RCC -> PLLCFGR |= (1U << 29);// SET N
 RCC -> PLLCFGR &= ~(7U << 4);
 RCC -> PLLCFGR &= ~(127U << 8);
 RCC -> PLLCFGR |= (8U << 8);//SET R
 GPIOB -> MODER &= ~(3U << 2 * 3);
 GPIOB -> MODER |= (2U << 2 * 3); // making pb3 alterna mode
 // PB3 AF2 timer2 chanel 2
 GPIOB -> AFR[0] &= ~(0xFU << 4 * 3);
 GPIOB -> AFR[0] |= (2U << 4 * 3); // AFSEL11 to AF2 0010
 RCC -> APBENR1 |= (1U << 0); //timer 2 clock open
 TIM2 -> CR1 = 0; // reset
 TIM2 -> CR1 |= (1U << 7); //arpe
 TIM2 -> PSC = 0; // set zero
 TIM2 -> ARR =64;
 TIM2 -> DIER |= (1U << 0);
 TIM2 -> CNT = 4294967295; //max value
 TIM2 -> CCMR1 &= ~(0x7U << 12); // clear
 TIM2 -> CCMR1 &= ~(0x1U << 24); // clean
 TIM2 -> CCMR1 |= (0x6U << 12); //0110
 TIM2 -> CCMR1 |= (1U << 11); //preload enable
 TIM2 -> CCER |= (1U << 4); //cmpr2 enable
 duty = 0;
 TIM2 -> CCR2 = duty;
 TIM2 -> CR1 |= (1U << 0); //enable tim2
 NVIC_SetPriority(TIM2_IRQn, 0); //setting priority and enabling
 NVIC_EnableIRQ(TIM2_IRQn);
}
void init_timer1() {
 RCC -> APBENR2 |= (1U << 11);
 TIM1 -> CR1 = 0;
 TIM1 -> CR1 |= (1 << 7);
 TIM1 -> CNT = 0;
 TIM1 -> PSC = 99;
 TIM1 -> ARR = 999;
 TIM1 -> DIER |= (1 << 0);
 TIM1 -> CR1 |= (1 << 0);
 NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0);
 NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
}
int counter =0 ;


void ModeSelect(){
	// Square wave
	if(mode==2){
		if(counter==0){
				duty=500; counter=1;
			}
			else if(counter=1){
				duty=0; counter=0;
			}
	}
	// Triangler wave
	else if(mode==3){
		 if(counter == 0 ){
				duty = duty +1;

			  if ( duty == 50){ counter =1;}
				}
				else if (counter ==1) {

				duty = duty -1;
				if (duty == 0) counter =0;}
```

```
}
// Sine wave
else if(mode==1){
            if(counter == 0 ){
                        duty = duty +3;
                        if ( duty == 24){ counter =1;}
            }
            else if(counter ==1) {
                         duty = duty +2;
                         if (duty == 50) {counter =2;}
            }
            else if(counter ==2) {
                         duty = duty +1;
                         if (duty == 75){ counter =3;}
            }
            else if(counter ==3) {
                         duty = duty + 1;
                         if (duty == 100){ counter =4;}
            }
            else if(counter ==4){
                         duty = duty - 1;
                         if(duty<75){counter =5;}
            }
            else if(counter ==5){
                         duty = duty-1;
                         if(duty<50){counter =6;}
            }
            else if(counter ==6){
                         duty = duty-2;
                         if(duty<24){counter =7;}
            }
            else if(counter ==7){
                         duty= duty-3;
                         if(duty==0){counter=0;}
            }
}
// Sawtooth wave
else if(mode==4){
            if(counter == 0){
                         duty= duty+1;
                         if(duty == 9900){ counter =1;}}
             else if(counter ==1){ duty=0;
                         if (duty==0){counter=0;}
              }
            }
            // White noise wave
            else if(mode==5){
                        if(counter == 0 ){
                                    duty= +99;
                                    if(duty>100){counter=1;}}
                              else if(counter==1){ duty=-198;
                                    if(duty>=100){counter=0; }}

            }}
void TIM1_BRK_UP_TRG_COM_IRQHandler(void) {
            ModeSelect();
            TIM2 -> CCR2 = duty;
            TIM1 -> SR &= ~(1U << 0);
}
void TIM2_IRQHandler(void) {

 TIM2 -> SR &= ~(1U << 0);
}
```

Conclusion

Within the scope of this project, the working principles of SSD with 4x4 keypad have been learned and integrated works have been provided. Using of interrupt and timer on coding has been learned and implemented. It was learned that a timed structure can be established with the timer. It has been seen that even very complex systems, a separate structure can be addedto the code by adding. Datasheet reading and component initilazing have been learned. The result is obtained with the set-reset combinations of the components. We have successfully designed a function generator within the scope of the project. With this generator we designed, we saw various values on the oscilloscope. Our project can work in six different modes as it wishes. It is also in use for our values in the analog rc filter we designed.

References

Okelo,” https://theokelo.co.ke/how-to-get-your-hs420361k-32-4-digit-7-segment-display-working-with-an-arduino/

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

STM32G0x1 Reference Manual “https://www.st.com/resource/en/reference_manual/dm00371828-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Gebze Teknik Üniversitesi Electric Circuits Lab