GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC 335

PROJECT 1

CONTENTS

1 PROBLEM

1.1 DEFINITION

This project is designed function generator. Different forms of pwm signals were produced and the desired signals were observed on the oscilloscope screen with the help of a second-order low-pass filter. # is used as enter and * is used as dot. While the A key on the keypad is used to enter the amplitude value, the B key is used to enter the frequency value. **If no key is pressed for 10 seconds, the operation is canceled due to timeout.** Button C will cycle through modes. Button D will cycle through displaying the Mode, Amplitude and Frequency of the currently active signal. **In addition, if the system presses a key and presses another key without entering this value, the value entered for that key is also cancelled.**

● There is no bouncing on the buttons. (A 25ms delay has been added to the input of interrupt functions.)

● There is no brightness difference between Seven Segments.

● There is no flickering on the SSDs.

2 AMPLITUDE

The amplitude value is entered by pressing the A key. The way the A key works is given below. **(It is seen that the point is entered by pressing the * key.)**

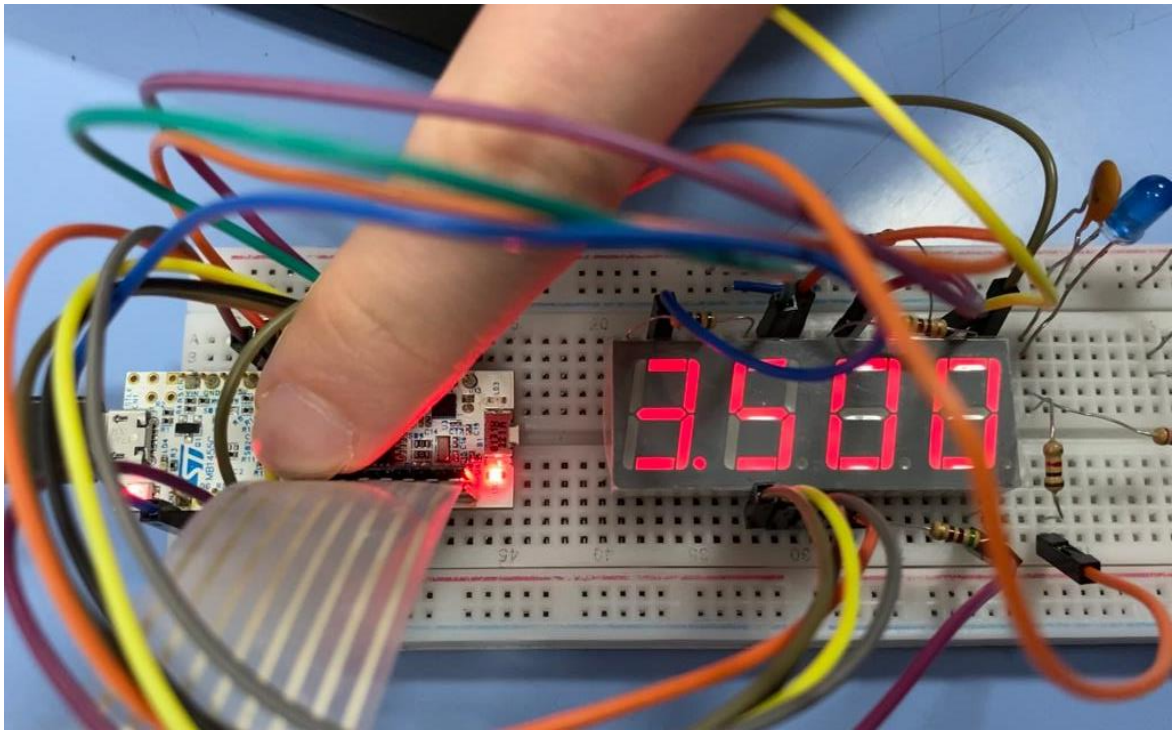

Figure 1: 3.5 V (Amplitude)

3 FREQUENCY

The frequency value is entered by pressing the B key. The way the B key works is given below. **(It is seen that the point is entered by pressing the * key.)**

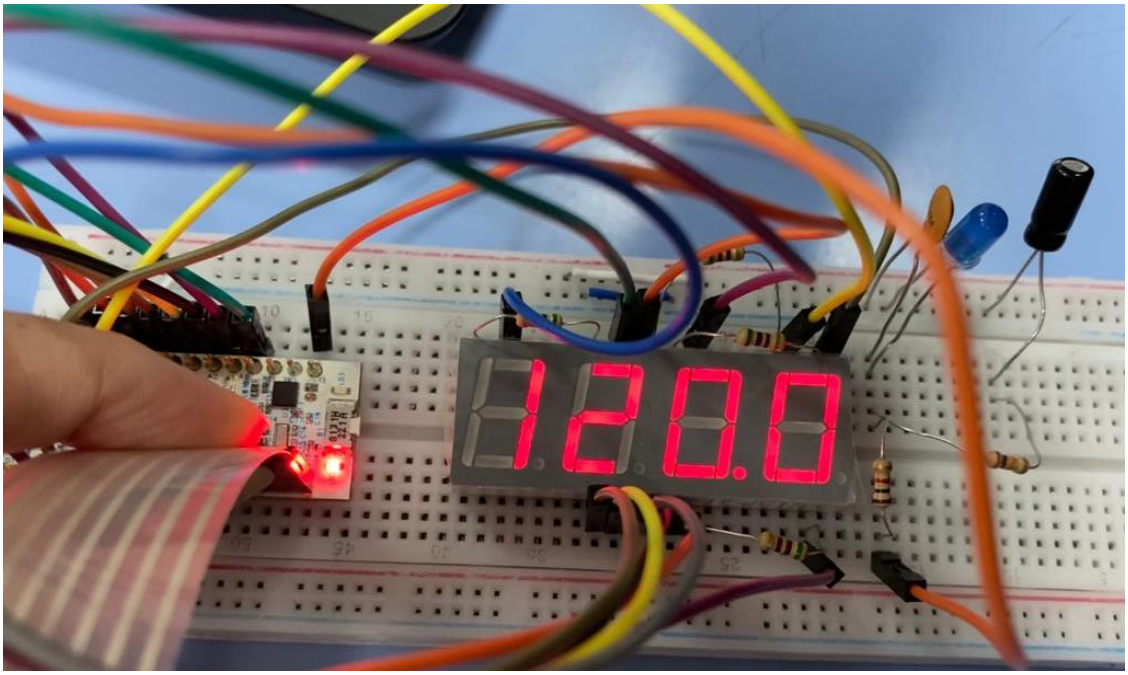

Figure 2: 120Hz

4 C AND D KEYS

As stated in the description, the C key will toggle between cycle through modes, while the d key will toggle between displaying the current display's mode, amplitude and frequency. How this is done is given below.

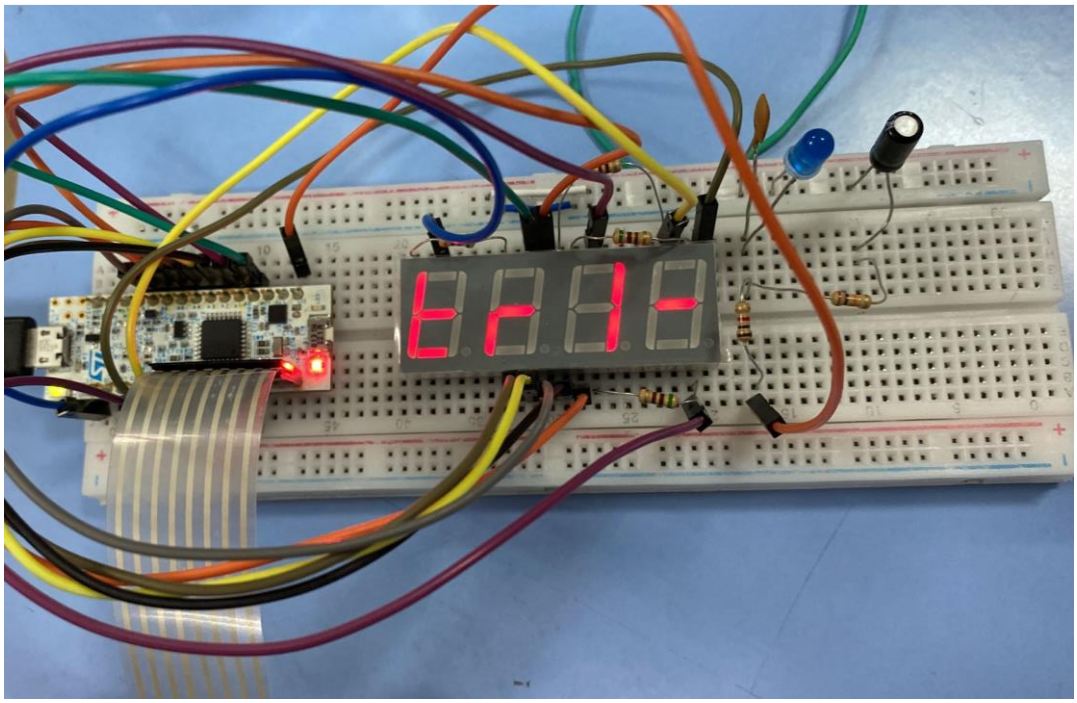

Figure 3: Triangular

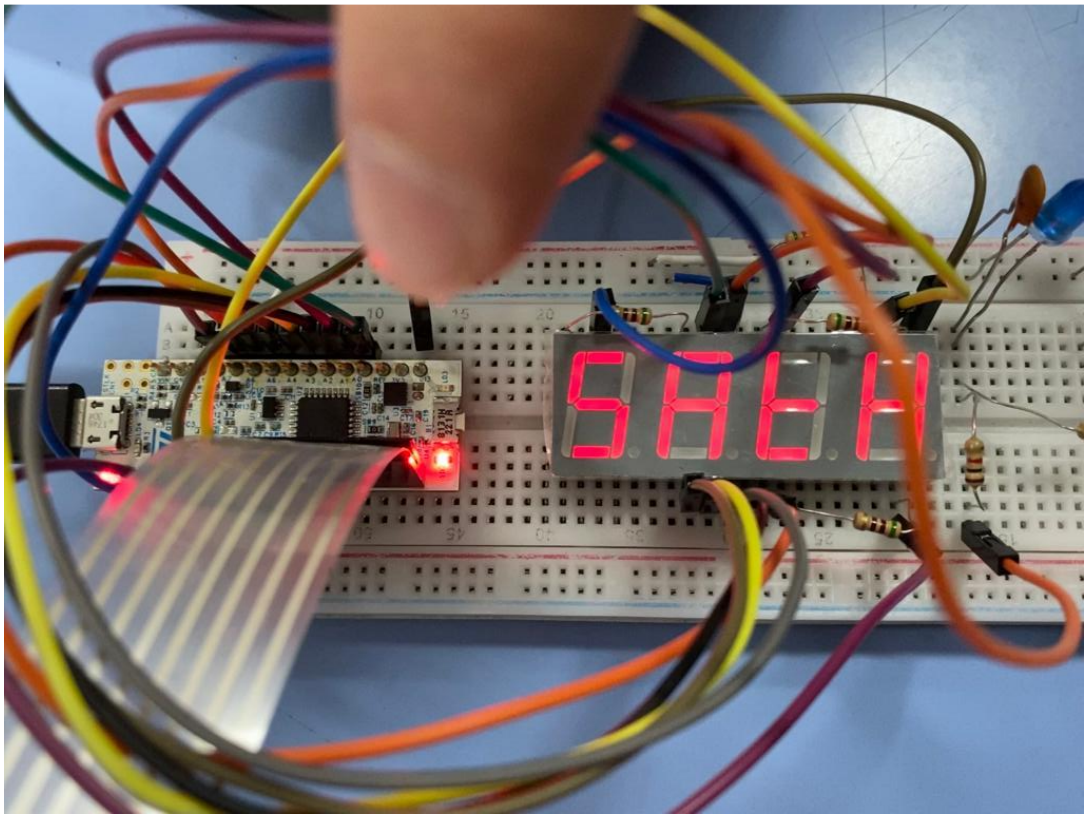

Figure 4: SawTooth

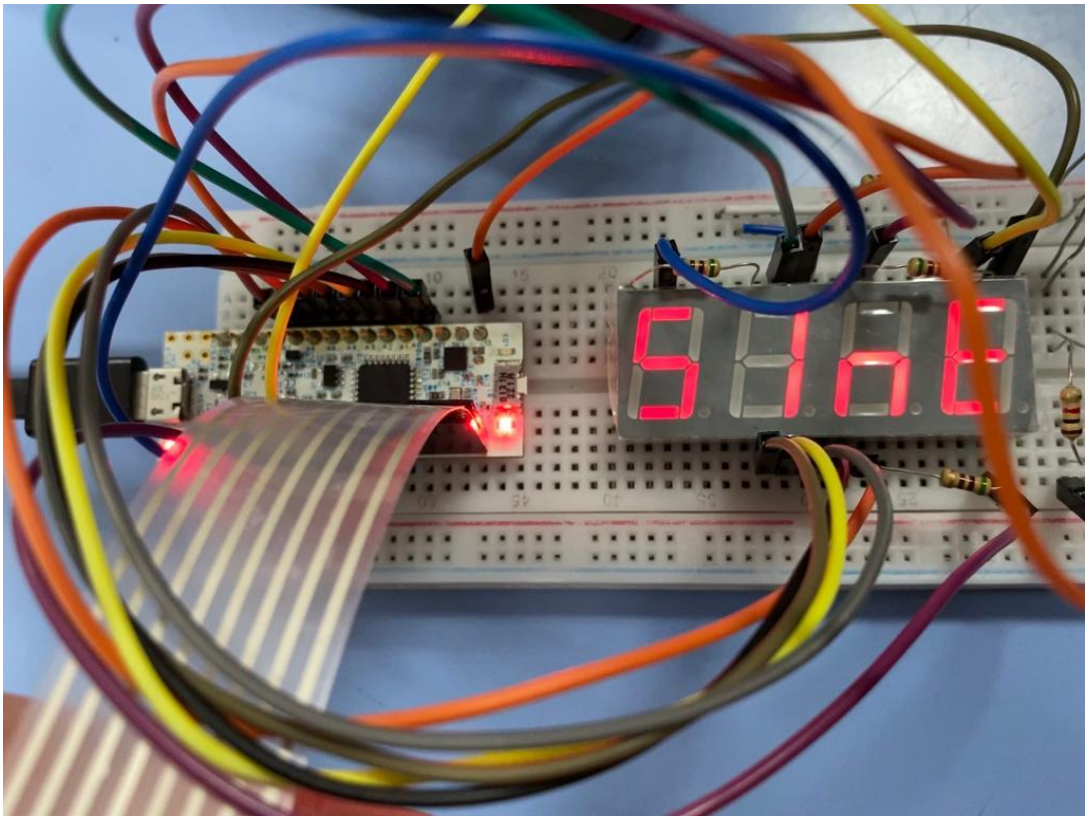

Figure 5: Sine

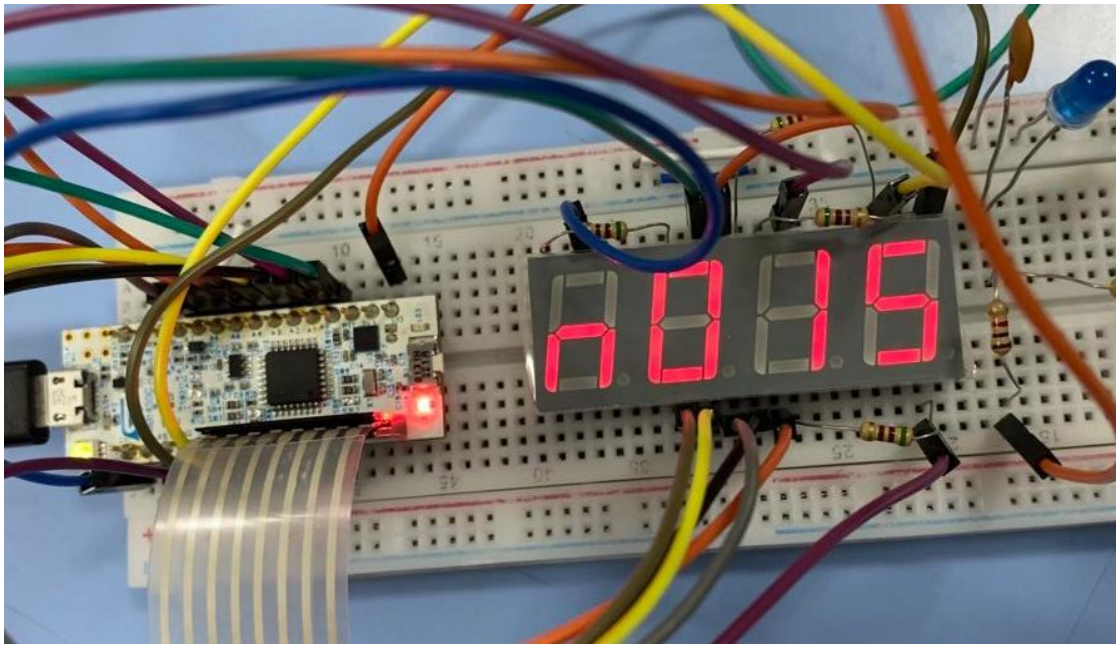

Figure 6: Noise

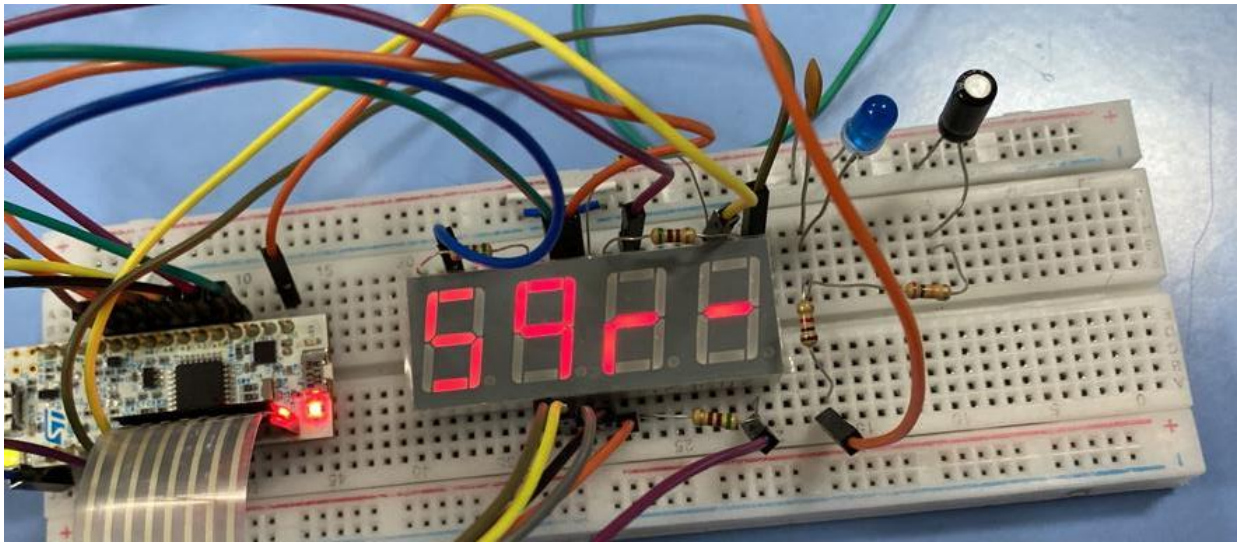

Figure 7: Square

5 LOW PASS FILTER

A low-pass filter is designed to get output from the oscilloscope. While making this design, the frequency value was taken into consideration and LTspice was used as the simulation program.

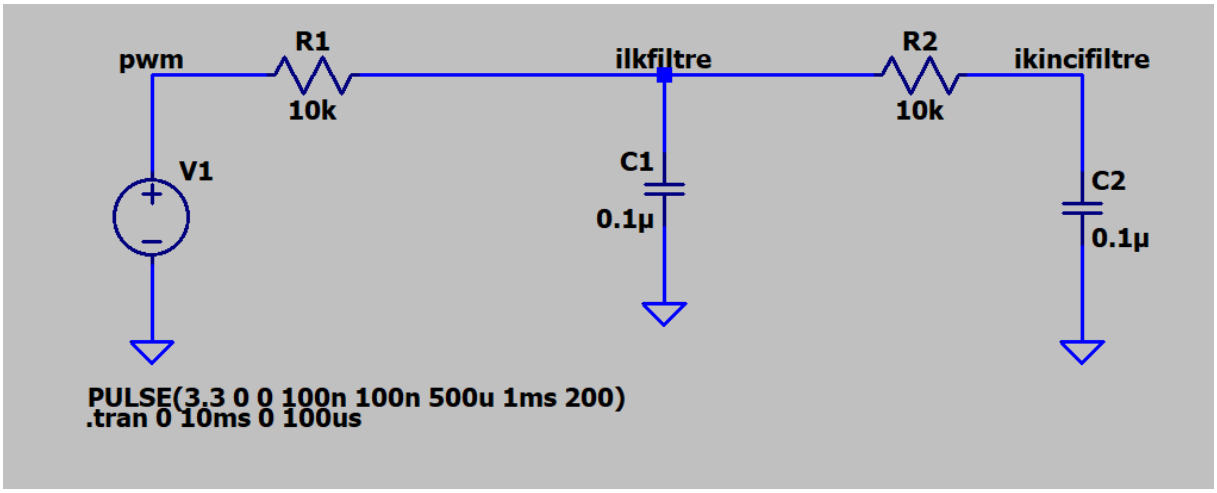

Figure 8: Low Pass Filter For PWM Signal

The outputs of the filter are as follows.

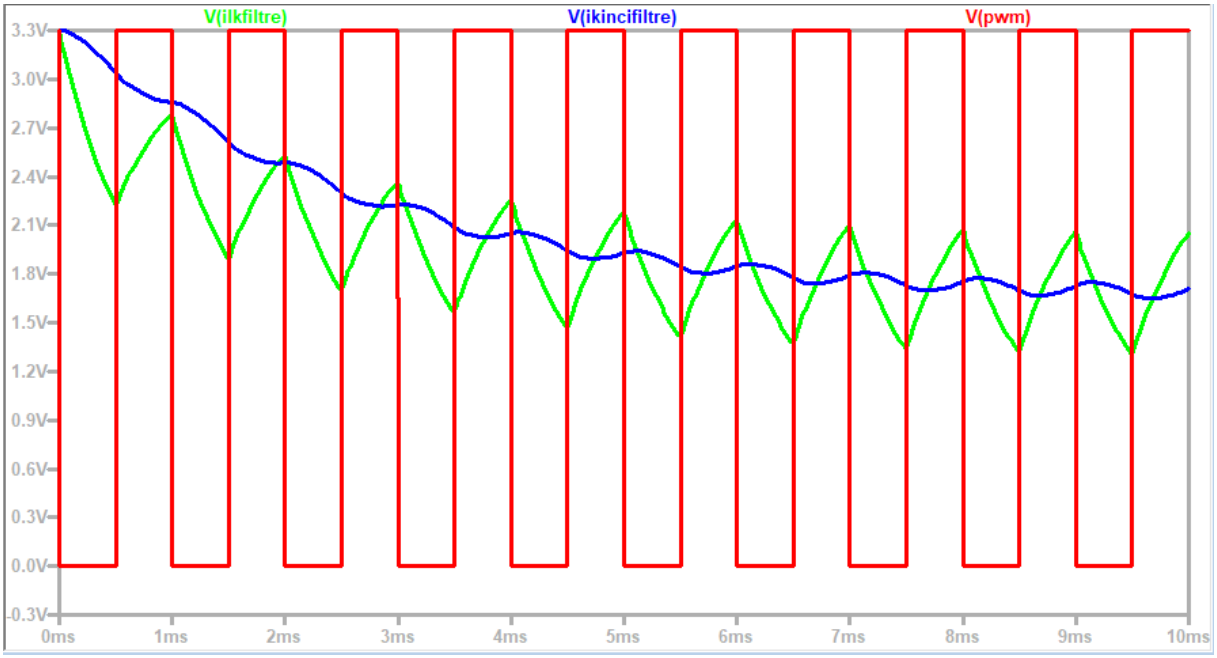

Figure 9: Low Pass Filter Output

6 OSCILLOSCOPE

When the second order filter is used, the desired graphics are obtained on the oscilloscope screen. Special functions have been written for all models such as square wave, sine, and digital stream output. As an example, an array is drawn from matlab for **sine.** While doing this, care has been taken to ensure that it has sufficient sensitivity for the sampling frequency in the size of the captured array and that it does not take up too much space. An array of 63 elements is considered sufficient. The function for which the sine is written is as follows.

```
void sine_Output(float amp, float freq){
	float periyot = 1/freq;
	periyot = periyot*1000;
	int i = 0;
	float k;
	k = (float) (2000*(amp/3.3));
	BSP_ms_timer1(100);
	while(timer1 != 0){
		TIM2->CCR2 = (uint32_t) (sinArray[i]*k);
		BSP_delay_ms((float) (periyot/63));
		i++;
		if(i>62) {i=0;}
	}
}
```

The oscilloscope image below is taken from the output. The waveform is triangular, the frequency is 5Hz, the amplitude is 2V, and the oscilloscope image is as follows.

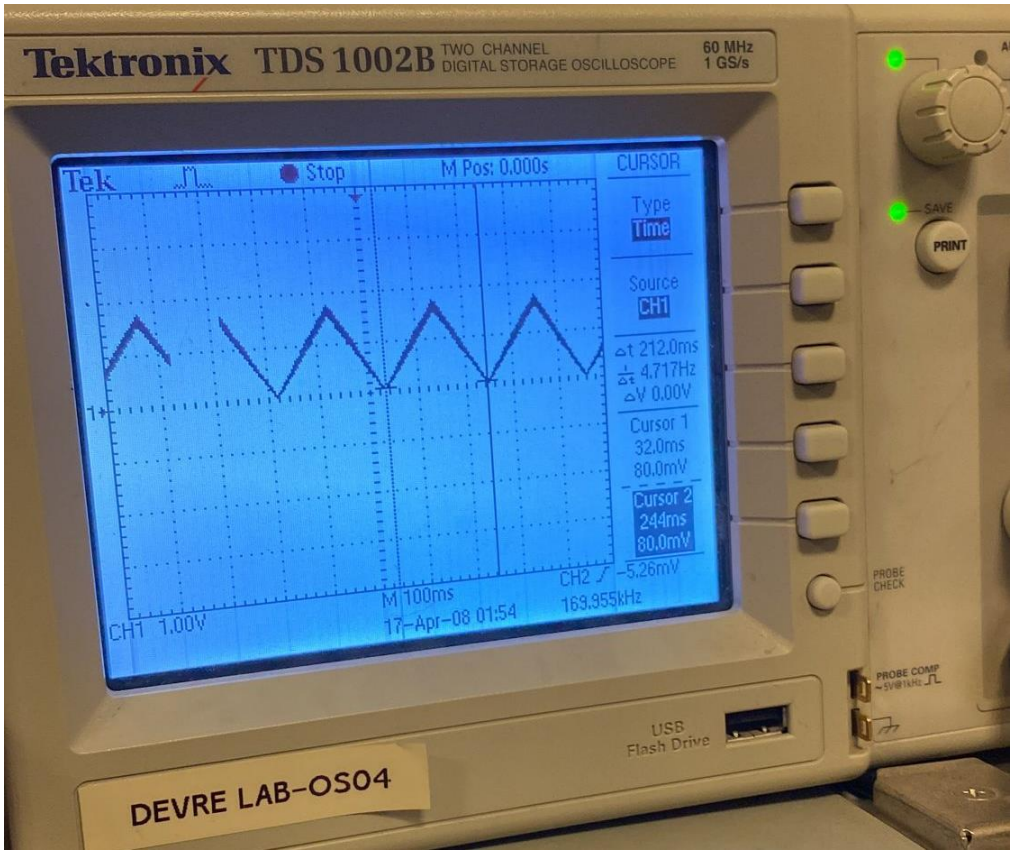

Figure 10: Triangular (F=5Hz, A=2V)

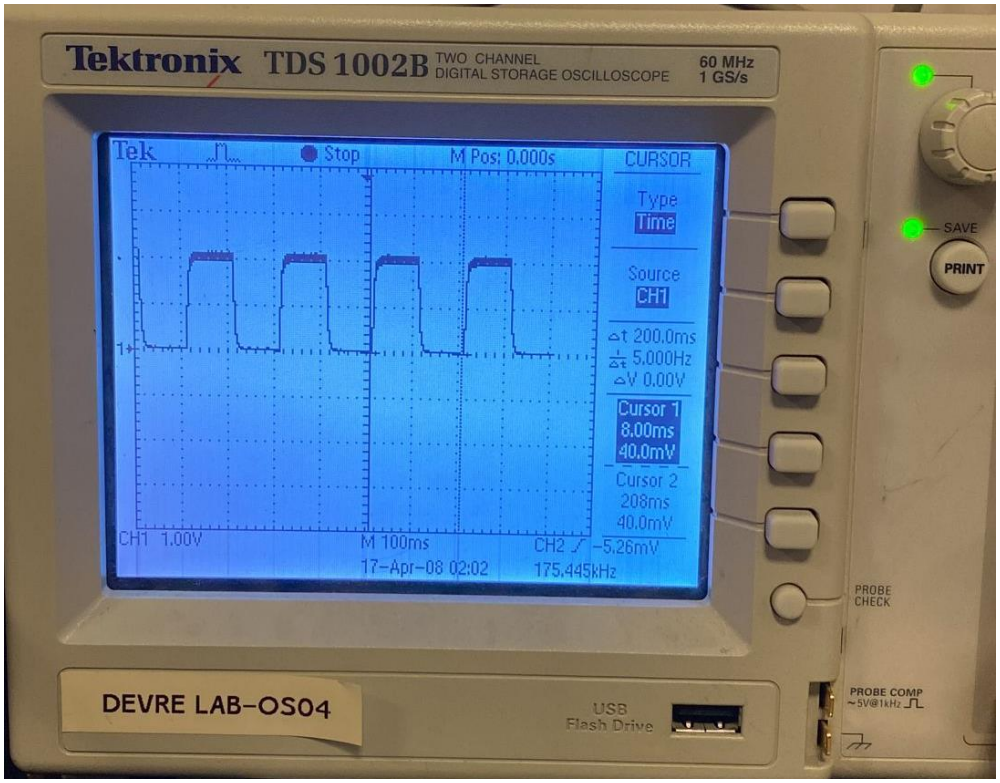

Figure 11: Square (F=5hZ,A=2V)

Figure 11 shows a **square** wave output. On the keypad, the amplitude value is set as 2V and the frequency value is set as 5Hz. **Exactly entered results are observed on the oscilloscope screen.**

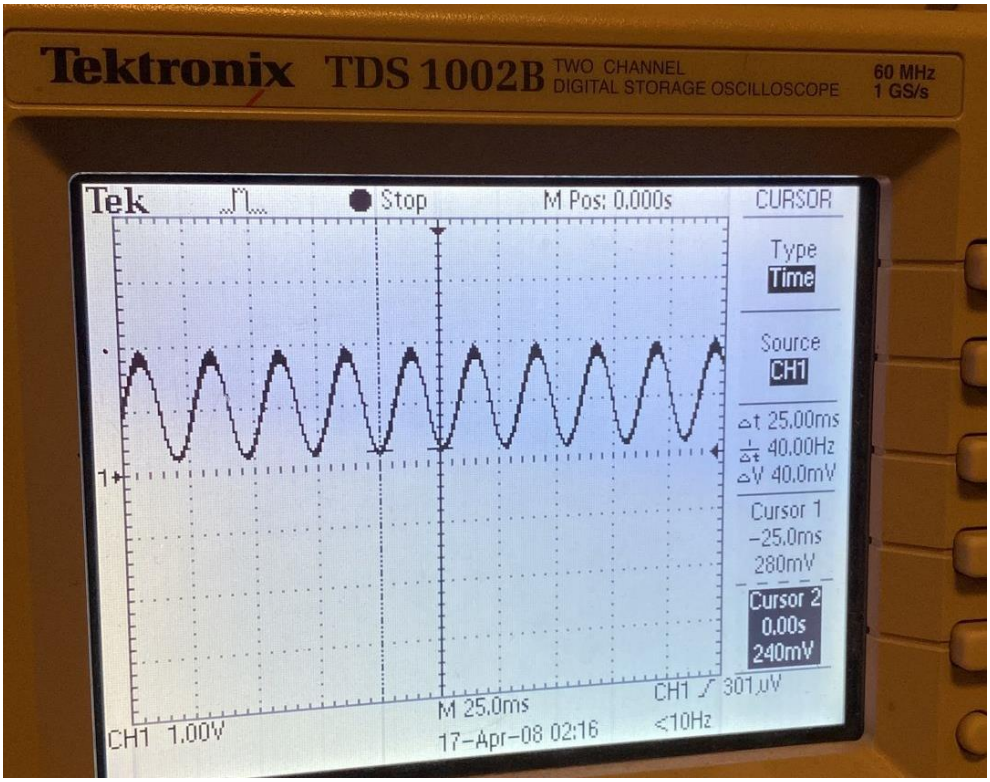

Figure 12: Sine (F=40 Hz, A=2V)

Figure 12 shows a **sine** wave output. On the keypad, the amplitude value is set as 2V and the frequency value is set as 40Hz. **Exactly entered results are observed on the oscilloscope screen.**

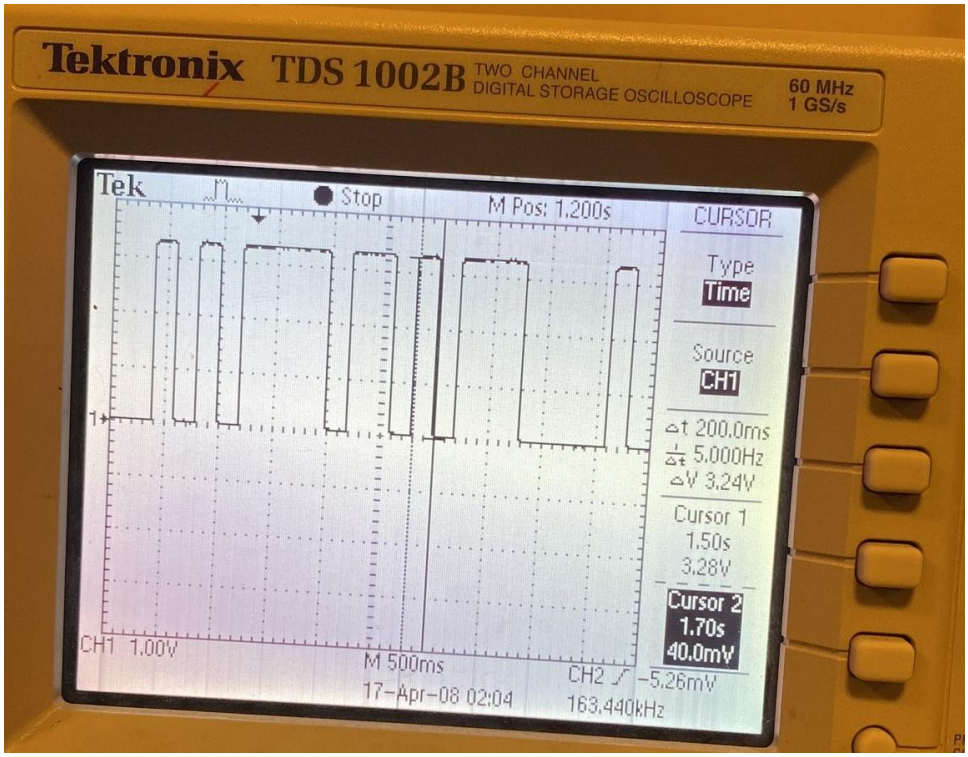

Figure 13: Digital Stream Output

In figure 13, the output is given as a **digital stream output signal.** The output for this mode is from a different pin than the others.

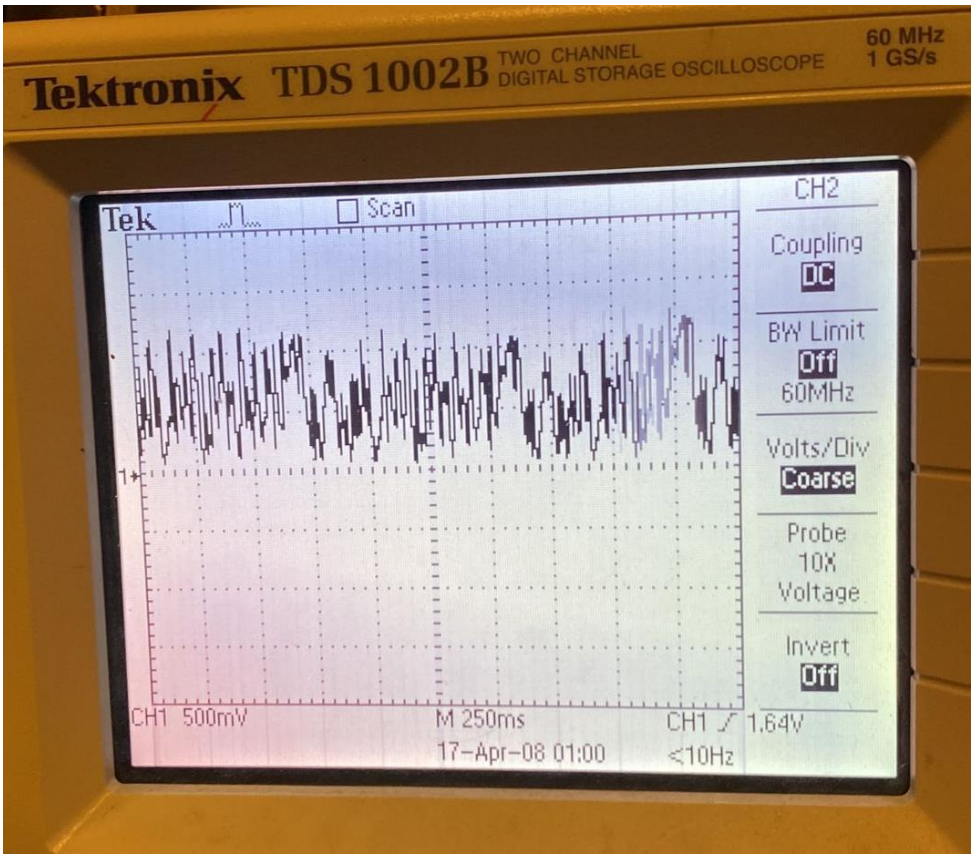

Figure 14: White Noise

Figure 14 shows **white noise.**

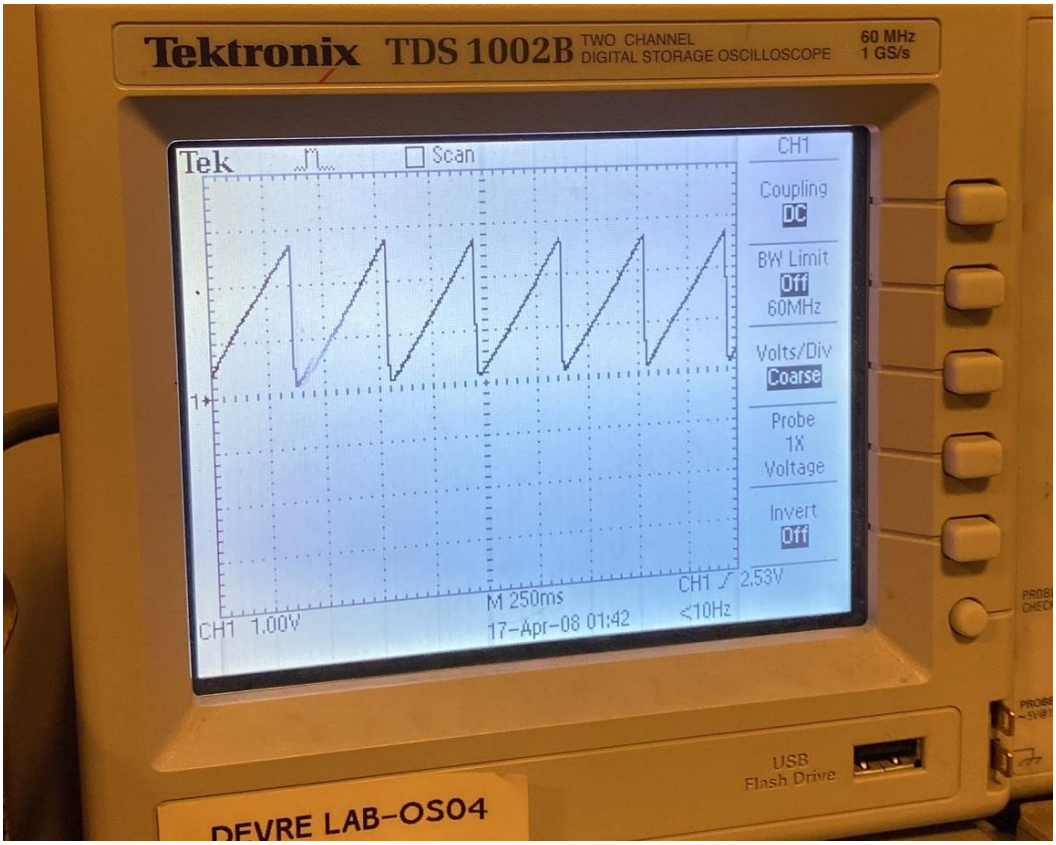

Figure 15: SawTooth

Sawtooth is given as the final output as in figure 15. It meets the entered frequency and amplitude values.

7 CODE

```
/*
 * bsp.c
 *
 *  Created on: 27 Kas 2021
 *      Author: Oğuzhan
 */

#include "bsp.h"

static volatile uint32_t tick = 0;
static volatile uint32_t timer1 = 0;
double randNumber[100] =
{0.7829,0.6938,0.0098,0.8432,0.9223,0.7710,0.0427,0.3782,0.7043,0.7295,0.2243,0.26
91,0.6730,0.4775,0.6237,0.2364,0.1771,0.8296,0.7669,0.9345,0.1079,0.1822,0.0991,0.
4898,0.1932,0.8959,0.0991,0.0442,0.5573,0.7725,0.3119,0.1790,0.3390,0.2101,0.5102,
0.9064,0.6289,0.1015,0.3909,0.0546,0.5013,0.4317,0.9976,0.8116,0.4857,0.8944,0.137
5,0.3900,0.9274,0.9175,0.7136,0.6183,0.3433,0.9360,0.1248,0.7306,0.6465,0.8332,0.3
983,0.7498,0.8352,0.3225,0.5523,0.9791,0.5493,0.3304,0.6195,0.3606,0.7565,0.4139,0
.4923,0.6947,0.9727,0.3278,0.8378,0.7391,0.9542,0.0319,0.3569,0.6627,0.2815,0.2304
```

```
,0.7111,0.6246,0.5906,0.6604,0.0476,0.3488,0.4513,0.2409,0.7150,0.8562,0.2815,0.73
11,0.1378,0.8367,0.1386,0.5882,0.3662,0.8068};
double sinArray[63] =
{1.0000,1.0998,1.1987,1.2955,1.3894,1.4794,1.5646,1.6442,1.7174,1.7833,1.8415,1.89
12,1.9320,1.9636,1.9854,1.9975,1.9996,1.9917,1.9738,1.9463,1.9093,1.8632,1.8085,1.
7457,1.6755,1.5985,1.5155,1.4274,1.3350,1.2392,1.1411,1.0416,0.9416,0.8423,0.7445,
0.6492,0.5575,0.4702,0.3881,0.3122,0.2432,0.1817,0.1284,0.0838,0.0484,0.0225,0.006
3,0.0001,0.0038,0.0175,0.0411,0.0742,0.1165,0.1677,0.2272,0.2945,0.3687,0.4493,0.5
354,0.6261,0.7206,0.8178,0.9169};


void BSP_system_init(){
      __disable_irq();
      genlik = 1;
      frekans = 1;
      sayac = 0;
      signal = 1;
      mod = 3;
      for (int i; i < 5; i++){
            a[i]=0;
            b[i]=0;
            c[i]=0;
      }
      set_signal();

    SystemCoreClockUpdate();
    BSP_set_RCC();
    BSP_led_init();
    BSP_button_init();
    BSP_init_timer1();
    init_timer2();
    BSP_init_SSD();
    SysTick_Config(SystemCoreClock / 100000);
    BSP_init_keypad();
    dSO_init();
    __enable_irq();
}

void SysTick_Handler(void){
      if(tick > 0) {
            --tick;
      }
}

void BSP_delay_ms(float s){
      tick = (uint32_t) (s*100);
      while (tick);
}

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

void BSP_set_RCC(){
    /* Enable GPIO ABC clock */
    RCC->IOPENR |= (7U << 0);
    RCC->IOPENR |= (1U << 5);
}
```

```c
// LED related functions

void BSP_led_init(){ // PB1
    /* Setup PC6 as output */
    GPIOC->MODER &= ~(3U << 2*6);
    GPIOC->MODER |= (1U << 2*6);

    /* Turn off LED */
    GPIOC->BRR |= (1U << 6);
}

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

void BSP_led_clear(){
    GPIOC->BRR |= (1U << 6);
}

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

// Button related functions
void BSP_button_init(){
    GPIOF->MODER &= ~(3U << 2*2);
}

int BSP_button_read(){
    int b = (GPIOF->IDR >> 2) & 0X1;
    if (b) return 0;
    else return 1;
}

//Timer related functions
void BSP_init_timer1(){

    RCC->APBENR2 |= (1U << 11); //Enable tim1 module clock

    TIM1->CR1 = 0;
    TIM1->CR1 |= (1 << 7);
    TIM1->CNT = 0;


    //1 sec interrupt
    TIM1->PSC = 999;
    TIM1->ARR = 16000;

    TIM1->DIER |= (1 << 0);
    TIM1->CR1 |= (1 << 0);
    // Fonksiyon icinde yapilacak
    // TIM1->SR &= ~(1U << 0);

    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0);
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
}
void BSP_ms_timer1(uint32_t milisec){
      timer1 = milisec;
}
```

```
void TIM1_BRK_UP_TRG_COM_IRQHandler(void){
	if ((timer1 == 1) && (sayac == 0)){
		if (mod == 1){
			a[0] = y[0];
			a[1] = y[1];
			a[2] = y[2];
			a[3] = y[3];
			a[4] = y[4];
		}
		else if(mod == 2){
			b[0] = y[0];
			b[1] = y[1];
			b[2] = y[2];
			b[3] = y[3];
			b[4] = y[4];
		}
	}
	if (timer1 > 0){
		timer1--;
	}



    TIM1->SR &= ~(1U << 0);
}
//Timer2 related functions

void init_timer2(void){
	RCC->APBENR1 |= (1U << 0); // RCC_APBENR1_TIM2EN

	GPIOB->AFR[0] &= ~(1U << 12);
	GPIOB->AFR[0] |= (1U << 13);
	GPIOB->AFR[0] &= ~(1U << 14);
	GPIOB->AFR[0] &= ~(1U << 15);

	// Set alternate function to 2
	// 3 comes from PB3
	// GPIOB->AFR[0] |= (2U << 4*3);

	//PB3 Alternate
	GPIOB->MODER |= (1U << 7);
	GPIOB->MODER &= ~(1U << 6);

	TIM2->CR1 = 0; //control register
//	TIM2->CR1 |= (1U << 7); //ARPE

	// Select PWM Mode 1
	TIM2->CCMR1 |= (6U << 12);
	// Preload Enable
	TIM2->CCMR1 |= TIM_CCMR1_OC2PE;

//	TIM2->CCMR1 |= (6U << 12);
//	TIM2->CCMR1 |= (1U << 11);
	// Capture compare ch2 enable
	TIM2->CCER |= TIM_CCER_CC2E;

	// zero out counter
	TIM2->CNT = 0;
   // 1 ms interrupt
```

```
    TIM2->PSC = 3;
    TIM2->ARR = 4000;

    // zero out duty
    TIM2->CCR2 = 0;

    // Update interrupt enable
    TIM2->DIER |= (1 << 0);
    // TIM1 Enable
    TIM2->CR1 |= TIM_CR1_CEN;

    NVIC_SetPriority(TIM2_IRQn, 1);
    NVIC_EnableIRQ(TIM2_IRQn);
}
void TIM2_IRQHandler(void){
      TIM2->SR &= ~(1U << 0);
}


//SSD related functions

void BSP_init_SSD(){

//Setup PA0, PA1, PA4, PA5, PA12, PA11, PA6 as output for SSD and Set PB6 , PA10,
PB1, PA15 as output for dp of the SSD
    GPIOA->MODER &= ~(3U << 2*0);
    GPIOA->MODER |= (1U << 2*0);

    GPIOA->MODER &= ~(3U << 2*1);
    GPIOA->MODER |= (1U << 2*1);

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

    GPIOA->MODER &= ~(3U << 2*5);
    GPIOA->MODER |= (1U << 2*5);

    GPIOA->MODER &= ~(3U << 2*12);
    GPIOA->MODER |= (1U << 2*12);

    GPIOA->MODER &= ~(3U << 2*11);
    GPIOA->MODER |= (1U << 2*11);

    GPIOA->MODER &= ~(3U << 2*6);
    GPIOA->MODER |= (1U << 2*6);

    GPIOA->MODER &= ~(3U << 2*7);
    GPIOA->MODER |= (1U << 2*7);

    GPIOB->MODER &= ~(3U << 2*6);
    GPIOB->MODER |= (1U << 2*6);

    GPIOA->MODER &= ~(3U << 2*10);
    GPIOA->MODER |= (1U << 2*10);

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

    GPIOA->MODER &= ~(3U << 2*15);
```

```
    GPIOA->MODER |= (1U << 2*15);

    BSP_clear_SSD();
}

void BSP_set_SSD(int x, int dx, int dpx){
      BSP_clear_SSD();
      switch(dx){
      case 1:
            GPIOB->BRR |= (1U << 6); // Set dp1(PB6) = 0
            if (1 == dpx){
                  GPIOA->ODR |= (1U << 7);
            }
            break;
      case 2:
            GPIOA->BRR |= (1U << 10); // Set dp2(PA10) = 0
            if (2 == dpx){
                  GPIOA->ODR |= (1U << 7);
            }
            break;
      case 3:
            GPIOB->BRR |= (1U << 1); // Set dp3(PB1) = 0
            if (3 == dpx){
                  GPIOA->ODR |= (1U << 7);
            }
            break;
      case 4:
            GPIOA->BRR |= (1U << 15); // Set dp4(PA15) = 0
            break;
      }
      switch(x) {
      case 0:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 11); // F
            BSP_delay_ms(6);
            break;
      case 1:
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            BSP_delay_ms(2);
            break;
      case 2:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 12); // E
            GPIOA->ODR |= (1U << 6); // G
            BSP_delay_ms(5);
            break;
      case 3:
            GPIOA->ODR |= (1U << 0); // A
            GPIOA->ODR |= (1U << 1); // B
            GPIOA->ODR |= (1U << 4); // C
            GPIOA->ODR |= (1U << 5); // D
            GPIOA->ODR |= (1U << 6); // G
```

```
      BSP_delay_ms(5);
      break;
case 4:
      GPIOA->ODR |= (1U << 1); // B
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(4);
      break;
case 5:
      GPIOA->ODR |= (1U << 0); // A
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 5); // D
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(5);
      break;
case 6:
      GPIOA->ODR |= (1U << 0); // A
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 5); // D
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(6);
      break;
case 7:
      GPIOA->ODR |= (1U << 0); // A
      GPIOA->ODR |= (1U << 1); // B
      GPIOA->ODR |= (1U << 4); // C
      BSP_delay_ms(3);
      break;
case 8:
      GPIOA->ODR |= (1U << 0); // A
      GPIOA->ODR |= (1U << 1); // B
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 5); // D
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(7);
      break;
case 9:
      GPIOA->ODR |= (1U << 0); // A
      GPIOA->ODR |= (1U << 1); // B
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 5); // D
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(6);
      break;
case 10: // A
      GPIOA->ODR |= (1U << 0); // A
      GPIOA->ODR |= (1U << 1); // B
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(6);
```

```
      break;
case 11: // B
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 5); // D
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(5);
      break;
case 12: // C
      GPIOA->ODR |= (1U << 0); // A
      GPIOA->ODR |= (1U << 5); // D
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 11); // F
      BSP_delay_ms(4);
      break;
case 13: // D
      GPIOA->ODR |= (1U << 1); // B
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 5); // D
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(5);
      break;
case 14: // *
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(1);
      break;
case 15: // #
      GPIOA->ODR |= (1U << 1); // B
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(5);
      break;
case 16: //E
      GPIOA->ODR |= (1U << 0); // A
      GPIOA->ODR |= (1U << 5); // D
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(5);
      break;
case 17: //n
      GPIOA->ODR |= (1U << 4); // C
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(3);
      break;
case 18: //t
      GPIOA->ODR |= (1U << 5); // D
      GPIOA->ODR |= (1U << 12); // E
      GPIOA->ODR |= (1U << 11); // F
      GPIOA->ODR |= (1U << 6); // G
      BSP_delay_ms(4);
      break;
case 19: //r
      GPIOA->ODR |= (1U << 12); // E
```

```
			GPIOA->ODR |= (1U << 6); // G
			BSP_delay_ms(2);
			break;
		case 20: // -
			GPIOA->ODR |= (1U << 6); // G
			BSP_delay_ms(1);
			break;
		case 21: //q
			GPIOA->ODR |= (1U << 0); // A
			GPIOA->ODR |= (1U << 1); // B
			GPIOA->ODR |= (1U << 4); // C
			GPIOA->ODR |= (1U << 11); // F
			GPIOA->ODR |= (1U << 6); // G
			BSP_delay_ms(5);
			break;
		case 22: //h
			GPIOA->ODR |= (1U << 1); // B
			GPIOA->ODR |= (1U << 4); // C
			GPIOA->ODR |= (1U << 12); // E
			GPIOA->ODR |= (1U << 11); // F
			GPIOA->ODR |= (1U << 6); // G
			BSP_delay_ms(5);
			break;
		}
}

// clear PA0, PA1, PA4, PA5, PA12, PA11, PA6 for SSD and Set PB6, PA10, PB1, PA15
for dp of SSD
void BSP_clear_SSD(){
		GPIOA->BRR |= (1U << 0); // A
		GPIOA->BRR |= (1U << 1); // B
		GPIOA->BRR |= (1U << 4); // C
		GPIOA->BRR |= (1U << 5); // D
		GPIOA->BRR |= (1U << 12); // E
		GPIOA->BRR |= (1U << 11); // F
		GPIOA->BRR |= (1U << 6); // G
		GPIOA->BRR |= (1U << 7); // DP
		GPIOB->ODR |= (1U << 6); // dp(PB6)
		GPIOA->ODR |= (1U << 10); // dp(PA10)
		GPIOB->ODR |= (1U << 1); // dp(PB1)
		GPIOA->ODR |= (1U << 15); // dp(PA15)
}

void BSP_SSD(int s1, int s2, int s3, int s4, int sd){
		BSP_set_SSD(s1,1,sd);
		BSP_set_SSD(s2,2,sd);
		BSP_set_SSD(s3,3,sd);
		BSP_set_SSD(s4,4,sd);
}

//Keypad related functions
void BSP_init_keypad(){
		/* Setup PA8(R4), PB9(R3), PB5(R2) and PB4(R1) as output */
    GPIOB->MODER &= ~(3U << 2*4);
    GPIOB->MODER |= (1U << 2*4);

    GPIOB->MODER &= ~(3U << 2*5);
    GPIOB->MODER |= (1U << 2*5);
```

```
    GPIOB->MODER &= ~(3U << 2*9);
    GPIOB->MODER |= (1U << 2*9);

    GPIOA->MODER &= ~(3U << 2*8);
    GPIOA->MODER |= (1U << 2*8);

    /* Setup PA9(C4), PB0(C3), PB2(C2) and PB8(C1) as input */
    GPIOB->MODER &= ~(3U << 2*8);
    GPIOB->PUPDR |= (2U << 2*8);

    GPIOB->MODER &= ~(3U << 2*2);
    GPIOB->PUPDR |= (2U << 2*2);

    GPIOB->MODER &= ~(3U << 2*0);
    GPIOB->PUPDR |= (2U << 2*0);

    GPIOA->MODER &= ~(3U << 2*9);
    GPIOA->PUPDR |= (2U << 2*9);

    /* Setup interrupts for inputs */
    EXTI->EXTICR[2] |= (0U << 8*1); // PA9
    EXTI->EXTICR[0] |= (1U << 8*0); // PB0
    EXTI->EXTICR[0] |= (1U << 8*2); // PB2
    EXTI->EXTICR[2] |= (1U << 8*0); // PB8

    /* Rising edge */
    EXTI->RTSR1 |= (1U << 9); // 9th pin
    EXTI->RTSR1 |= (1U << 0); // 0
    EXTI->RTSR1 |= (1U << 2);
    EXTI->RTSR1 |= (1U << 8);

    /* Mask */
    EXTI->IMR1 |= (1U << 9);
    EXTI->IMR1 |= (1U << 0);
    EXTI->IMR1 |= (1U << 2);
    EXTI->IMR1 |= (1U << 8);

    /* NVIC */
    NVIC_SetPriority(EXTI0_1_IRQn , 1);
    NVIC_EnableIRQ(EXTI0_1_IRQn);

    NVIC_SetPriority(EXTI2_3_IRQn , 1);
    NVIC_EnableIRQ(EXTI2_3_IRQn);

    NVIC_SetPriority(EXTI4_15_IRQn , 1);
    NVIC_EnableIRQ(EXTI4_15_IRQn);
}
void clearRowsKeypad(void){
      GPIOA->BRR |= (1U << 8); // PA8
      GPIOB->BRR |= (1U << 9);
      GPIOB->BRR |= (1U << 5);
      GPIOB->BRR |= (1U << 4);
}
void setRowsKeypad(void){
       GPIOA->ODR |= (1U << 8); // PA8
       GPIOB->ODR |= (1U << 9);
       GPIOB->ODR |= (1U << 5);
       GPIOB->ODR |= (1U << 4);
}
```

```
void EXTI0_1_IRQHandler(void){ //PB0 (C3) # , 9 , 6 , 3
	clearRowsKeypad();

	GPIOA->ODR ^= (1U << 8); //PA8
	if ((GPIOB->IDR >> 0) & 1 ) { // read PB0
	//	BSP_set_SSD(15); #
		sayac = 5; // no input anymore
		timer1 = 0;
		if (mod == 3){
			flag = 1;
		}
	}
	GPIOA->ODR ^= (1U << 8); //PA8

	GPIOB->ODR ^= (1U << 9); //PB9
	if ((GPIOB->IDR >> 0) & 1 ) { // read PB0
	//	BSP_set_SSD(9);
		if (sayac < 4 ){
			if (mod == 1){
				a[sayac] = 9;
			}
			else if (mod == 2){
				b[sayac] = 9;
			}
		}
		sayac ++;
	}
	GPIOB->ODR ^= (1U << 9); //PB9

	GPIOB->ODR ^= (1U << 5); //PB5
	if ((GPIOB->IDR >> 0) & 1 ) { // read PB0
		if (sayac < 4 ){
			if (mod == 1){
				a[sayac] = 6;
			}
			else if (mod == 2){
				b[sayac] = 6;
			}
		}
		sayac ++;
	}
	GPIOB->ODR ^= (1U << 5); //PB5

	GPIOB->ODR ^= (1U << 4); //PB4
	if ((GPIOB->IDR >> 0) & 1 ) { // read PB0
	//	BSP_set_SSD(3);
		if (sayac < 4 ){
			if (mod == 1){
				a[sayac] = 3;
			}
			else if (mod == 2){
				b[sayac] = 3;
			}
		}
		sayac ++;

	}
	GPIOB->ODR ^= (1U << 4); //PB4
```

```
	setRowsKeypad();
	EXTI->RPR1 |= (1U << 0);

}

void EXTI2_3_IRQHandler(void){ //PB2 (C2)
	clearRowsKeypad();

	GPIOA->ODR ^= (1U << 8); //PA8
	if ((GPIOB->IDR >> 2) & 1 ) { // read PB0
	//	BSP_set_SSD(0);
		if (sayac < 4 ){
			if (mod == 1){
				a[sayac] = 0;
			}
			else if (mod == 2){
				b[sayac] = 0;
			}
		}
		sayac ++;

	}
	GPIOA->ODR ^= (1U << 8); //PA8

	GPIOB->ODR ^= (1U << 9); //PB9
	if ((GPIOB->IDR >> 2) & 1 ) { // read PB0
	//	BSP_set_SSD(8);
		if (sayac < 4 ){
			if (mod == 1){
				a[sayac] = 8;
			}
			else if (mod == 2){
				b[sayac] = 8;
			}
		}
		sayac ++;

	}
	GPIOB->ODR ^= (1U << 9); //PB9

	GPIOB->ODR ^= (1U << 5); //PB5
	if ((GPIOB->IDR >> 2) & 1 ) { // read PB0
	//	BSP_set_SSD(5);
		if (sayac < 4 ){
			if (mod == 1){
				a[sayac] = 5;
			}
			else if (mod == 2){
				b[sayac] = 5;
			}
		}
		sayac ++;

	}
	GPIOB->ODR ^= (1U << 5); //PB5

	GPIOB->ODR ^= (1U << 4); //PB4
	if ((GPIOB->IDR >> 2) & 1 ) { // read PB0
```

```
	//	BSP_set_SSD(2);
		if (sayac < 4 ){
			if (mod == 1){
				a[sayac] = 2;
			}
			else if (mod == 2){
				b[sayac] = 2;
			}
		}
		sayac ++;

	}
	GPIOB->ODR ^= (1U << 4); //PB4

	setRowsKeypad();
	EXTI->RPR1 |= (1U << 2);

}

void EXTI4_15_IRQHandler(void){ //PB8 (C1) and PA9 (C4)

	if ( (EXTI->RPR1 >> 8) & 1 ) {
		clearRowsKeypad();

		GPIOA->ODR ^= (1U << 8); //PA8
		if ((GPIOB->IDR >> 8) & 1 ) { // read PB0
		//	BSP_set_SSD(14);
			if (sayac < 4 ){
				if (mod == 1){
					a[4] = sayac; // dpa
				}
				else if (mod == 2){
					b[4] = sayac; // dpb
				}
			}

		}
		GPIOA->ODR ^= (1U << 8); //PA8

		GPIOB->ODR ^= (1U << 9); //PB9
		if ((GPIOB->IDR >> 8) & 1 ) { // read PB0
			//BSP_set_SSD(7);
			if (sayac < 4 ){
				if (mod == 1){
					a[sayac] = 7;
				}
				else if (mod == 2){
					b[sayac] = 7;
				}
			}
			sayac ++;

		}
		GPIOB->ODR ^= (1U << 9); //PB9

		GPIOB->ODR ^= (1U << 5); //PB5
		if ((GPIOB->IDR >> 8) & 1 ) { // read PB0
		//	BSP_set_SSD(4);
			if (sayac < 4 ){
```

```
                        if (mod == 1){
                                a[sayac] = 4;
                        }
                        else if (mod == 2){
                                b[sayac] = 4;
                        }
                }
                sayac ++;

        }
        GPIOB->ODR ^= (1U << 5); //PB5

        GPIOB->ODR ^= (1U << 4); //PB4
        if ((GPIOB->IDR >> 8) & 1 ) { // read PB0
        //      BSP_set_SSD(1);
                if (sayac < 4 ){
                        if (mod == 1){
                                a[sayac] = 1;
                        }
                        else if (mod == 2){
                                b[sayac] = 1;
                        }
                }
                sayac ++;

        }
        GPIOB->ODR ^= (1U << 4); //PB4


        setRowsKeypad();
        EXTI->RPR1 |= (1U << 8);
}
if ( (EXTI->RPR1 >> 9) & 1 ) {
        clearRowsKeypad();

        GPIOA->ODR ^= (1U << 8); //PA8
        if ((GPIOA->IDR >> 9) & 1 ) { // read PB0
        //      BSP_set_SSD(13);
                if (mod < 3){
                        mod++;
                }
                else {
                        mod = 1;
                }
                sayac = 5;
        }
        GPIOA->ODR ^= (1U << 8); //PA8

        GPIOB->ODR ^= (1U << 9); //PB9
        if ((GPIOA->IDR >> 9) & 1 ) { // read PB0
        //      BSP_set_SSD(12);
                if (signal < 6){
                        signal ++;
                }
                else{
                        signal = 1;
                }
                set_signal();
```

```
			}
			GPIOB->ODR ^= (1U << 9); //PB9

			GPIOB->ODR ^= (1U << 5); //PB5
			if ((GPIOA->IDR >> 9) & 1 ) { // read PB0
			//	BSP_set_SSD(11);
				if ((timer1 != 0) && (mod == 1)){
					a[0] = y[0];
					a[1] = y[1];
					a[2] = y[2];
					a[3] = y[3];
					a[4] = y[4];
				}
				BSP_ms_timer1(11);
				mod = 2;
				sayac = 0;
				for (int i=0; i<5 ; i++){
					y[i] = b[i];
					b[i] = 0;
				}
			}
			GPIOB->ODR ^= (1U << 5); //PB5

			GPIOB->ODR ^= (1U << 4); //PB4
			if ((GPIOA->IDR >> 9) & 1 ) { // read PB0
			// BSP_set_SSD(10);
				if ((timer1 != 0) && (mod == 2)){
					b[0] = y[0];
					b[1] = y[1];
					b[2] = y[2];
					b[3] = y[3];
					b[4] = y[4];
				}
				BSP_ms_timer1(11);
				mod = 1;
				sayac = 0;
				for (int i=0; i<5 ; i++){
					y[i] = a[i];
					a[i] = 0;
				}
			}
			GPIOB->ODR ^= (1U << 4); //PB4
			setRowsKeypad();
			EXTI->RPR1 |= (1U << 9);
		}

}

//Signal related functions
void set_signal(void){
		switch (signal){
		case 1: //sine
			c[0]=5;
			c[1]=1;
			c[2]=17;
			c[3]=16;
			c[4]=5;
			break;
		case 2: //square
```

```
			c[0]=5;
			c[1]=21;
			c[2]=19;
			c[3]=20;
			c[4]=5;
			break;
		case 3: //triangle - tri
			c[0]=18;
			c[1]=19;
			c[2]=1;
			c[3]=20;
			c[4]=5;
			break;
		case 4: // ramp sawtooth - sath
			c[0]=5;
			c[1]=10;
			c[2]=18;
			c[3]=22;
			c[4]=5;
			break;
		case 5: //white noise - nois
			c[0]=17;
			c[1]=0;
			c[2]=1;
			c[3]=5;
			c[4]=5;
			break;
		case 6: //random digital stream - rdso
			c[0]=19;
			c[1]=13;
			c[2]=5;
			c[3]=0;
			c[4]=5;
			break;
		}
}

//Signals
void kare_Dalga(float amp, float freq){
		BSP_ms_timer1(20);
		while(timer1 != 0){
			TIM2->CCR2 = 0;
			BSP_delay_ms((float) (1000*1/(2*freq)));
			TIM2->CCR2 = (uint32_t) (amp*4000/(3.3));
			BSP_delay_ms((float) (1000*1/(2*freq)));
		}
}

void ucgen_Dalga(float amp, float freq){
		amp = (float) (amp*4000/3.3);
		float interval = 10;
		float saniye = (float) (freq*2*amp);
		saniye = (float) (interval/saniye);
		float milisaniye = (saniye * 1000);
		int yon = 1;
		BSP_ms_timer1(20);
		while (timer1 != 0){
			if(yon == 1){
				TIM2->CCR2 = TIM2->CCR2 + (uint32_t) interval;
```

```
                if(amp <= (TIM2->CCR2)){
                        yon = 0;
                }
            }
            else if(yon == 0){
                TIM2->CCR2 = TIM2->CCR2 - (uint32_t) interval;
                if((TIM2->CCR2) <= 0){
                        yon = 1;
                }
            }
            BSP_delay_ms((float) milisaniye);
        }
}

void saw_Tooth(float amp, float freq){
        amp = (float) (amp*4000/3.3);
        float interval = 10;
        float saniye = (float) (freq*amp);
        saniye = (float) (interval/saniye);
        float milisaniye = saniye * 1000;
        BSP_ms_timer1(20);
        while (timer1 != 0){
            TIM2->CCR2 = TIM2->CCR2 + (uint32_t) interval;
            if((TIM2->CCR2) >= amp){
                TIM2->CCR2 = 0;
            }
            BSP_delay_ms((float) milisaniye);
        }
}

void white_Noise(float amp, float freq){
        amp = (float) (amp*4000/3.3);
        float periyot = 1/freq;
        periyot = periyot*1000;
        int i = 0;
        int random;
        BSP_ms_timer1(20);
        while (timer1 != 0){
            random = (int) (randNumber[i]*10000);
            TIM2->CCR2 = (uint32_t) ( random % (int) amp+1);
            BSP_delay_ms((float) periyot);
            i++;
            if(i>99) {i=0;}
        }

}
void dSO_init(void){ //PB7
        GPIOB->MODER &= ~(3U << 2*7);
        GPIOB->MODER |= (1U << 2*7);

        GPIOB->BRR |= (1U << 7);
}
void digital_Stream_Output(float freq){
        float periyot = 1/freq;
        periyot = periyot*1000;
        int i = 0;
        uint32_t random;
        BSP_ms_timer1(20);
        while (timer1 != 0){
```

```
			GPIOB->ODR &= ~(1U << 7);
			random = (uint32_t) (randNumber[i]*10000);
			random = ((uint32_t) ( random % 2 ));
			GPIOB->ODR |= (random << 7);
			BSP_delay_ms((float) periyot);
			i++;
			if(i>99) {i=0;}
		}
}
void sine_Output(float amp, float freq){
		float periyot = 1/freq;
		periyot = periyot*1000;
		int i = 0;
		float k;
		k = (float) (2000*(amp/3.3));
		BSP_ms_timer1(100);
		while(timer1 != 0){
			TIM2->CCR2 = (uint32_t) (sinArray[i]*k);
			BSP_delay_ms((float) (periyot/63));
			i++;
			if(i>62) {i=0;}
		}
}

void signal_Out(){
		set_Values();
		switch (signal){
			case 1: //sine
				sine_Output(genlik,frekans);
				break;
			case 2: //square
				kare_Dalga(genlik,frekans);
				break;
			case 3: //triangle - tri
				ucgen_Dalga(genlik,frekans);
				break;
			case 4: // ramp sawtooth - sath
				saw_Tooth(genlik,frekans);
				break;
			case 5: //white noise - nois
				white_Noise(genlik,frekans);
				break;
			case 6: //random digital stream - rdso
				digital_Stream_Output(frekans);
				break;
			default:
				break;
		}
}
void set_Values(void){
		switch(a[4]){
		case 1:
			genlik = (float)  (a[0]*1 + a[1]*0.1 + a[2]*0.01 + a[3]*0.001);
			break;
		case 2:
			genlik = (float) 3.3;
			break;
		case 3:
			genlik = (float) 3.3;
```

```
			break;
		case 4:
			genlik = (float) 3.3;
			break;
		default:
			genlik = 1;
			break;
		}
		switch(b[4]){
		case 1:
			frekans = (float) (b[0]*1 + b[1]*0.1 + b[2]*0.01 + b[3]*0.001);
			break;
		case 2:
			frekans = (float) (b[0]*10 + b[1]*1 + b[2]*0.1 + b[3]*0.01);
			break;
		case 3:
			frekans = (float) (b[0]*100 + b[1]*10 + b[2]*1 + b[3]*0.1);
			break;
		case 4:
			frekans = (float) (b[0]*1000 + b[1]*100 + b[2]*10 + b[3]*1);
			break;
		default:
			frekans = 2;
		break;
		}
}
```

BSP H

```
/*
 * bsp.h
 *
 *  Created on: 27 Kas 2021
 *      Author: Oğuzhan
 */

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

int a[5];
int b[5];
int c[5];
int y[5];
int sayac, signal, mod, flag;
float genlik, frekans;


void delay(volatile unsigned int);
void BSP_system_init();
void BSP_set_RCC();

//SysTick
void SysTick_Handler(void);
void BSP_delay_ms(float s);
```

```
//led related functions PC6
void BSP_led_init();
void BSP_led_set();
void BSP_led_clear();
void BSP_led_toggle();

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

//Timer related functions
void BSP_init_timer1();
void BSP_ms_timer1(uint32_t milisec);

//timer2 related functions
void init_timer2(void);

//SSD related functions
void BSP_init_SSD();
void BSP_set_SSD(int x, int dx, int dpx);
void BSP_clear_SSD();
void BSP_SSD(int s1, int s2, int s3, int s4, int sd);

//Keypad related functions
void BSP_init_keypad();
void clearRowsKeypad(void);
void setRowsKeypad(void);

//Signal related functions
void set_signal(void);
void kare_Dalga(float amp, float freq);
void ucgen_Dalga(float amp, float freq);
void saw_Tooth(float amp, float freq);
void white_Noise(float amp, float freq);
void dSO_init(void);
void digital_Stream_Output(float freq);
void sine_Output(float amp, float freq);
void signal_Out(void);
void set_Values(void);

#endif /* BSP_H_ */
```

MAIN C

```
/*
 * main.c
 *
 * author: Furkan Cayci
 *
 * description: Blinks 1 on-board LED at roughly 1 second intervals. system
 *   clock is running from HSI which is 16 Mhz. Delay function is just a simple
 *   counter so is not accurate and the delay time will change based on the
 *   optimization flags.
 */

#include "stm32g0xx.h"
#include "bsp.h"
```

```c
#define LEDDELAY    1600000U


int main(void) {

    BSP_system_init();

    while (1){
      if(mod == 1){
            BSP_SSD(a[0],a[1],a[2],a[3],a[4]);
      }
      else if(mod == 2) {
            BSP_SSD(b[0],b[1],b[2],b[3],b[4]);
      }
      else if(mod == 3){
            BSP_SSD(c[0],c[1],c[2],c[3],c[4]);
            if (flag == 1){
                  signal_Out();
                  flag = 0;
            }
      }
      else{
            mod = 3;
      }
    }

    return 0;
}
```

8 FLOWCHART

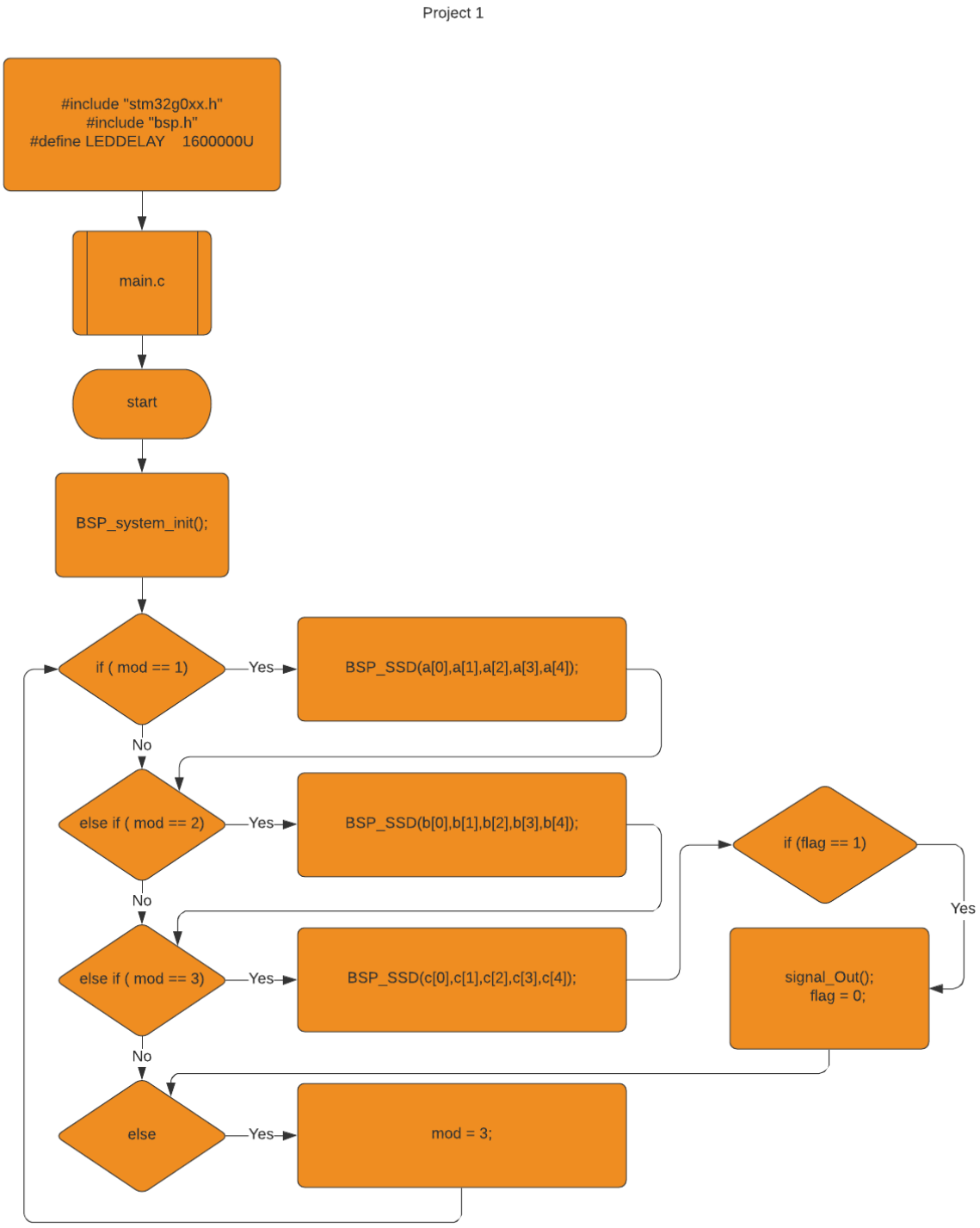

Figure 16: Flowchart of main function

Figure 17 : Flowchart of the header file(bsp.h)

Important functions from bsp.c for the project

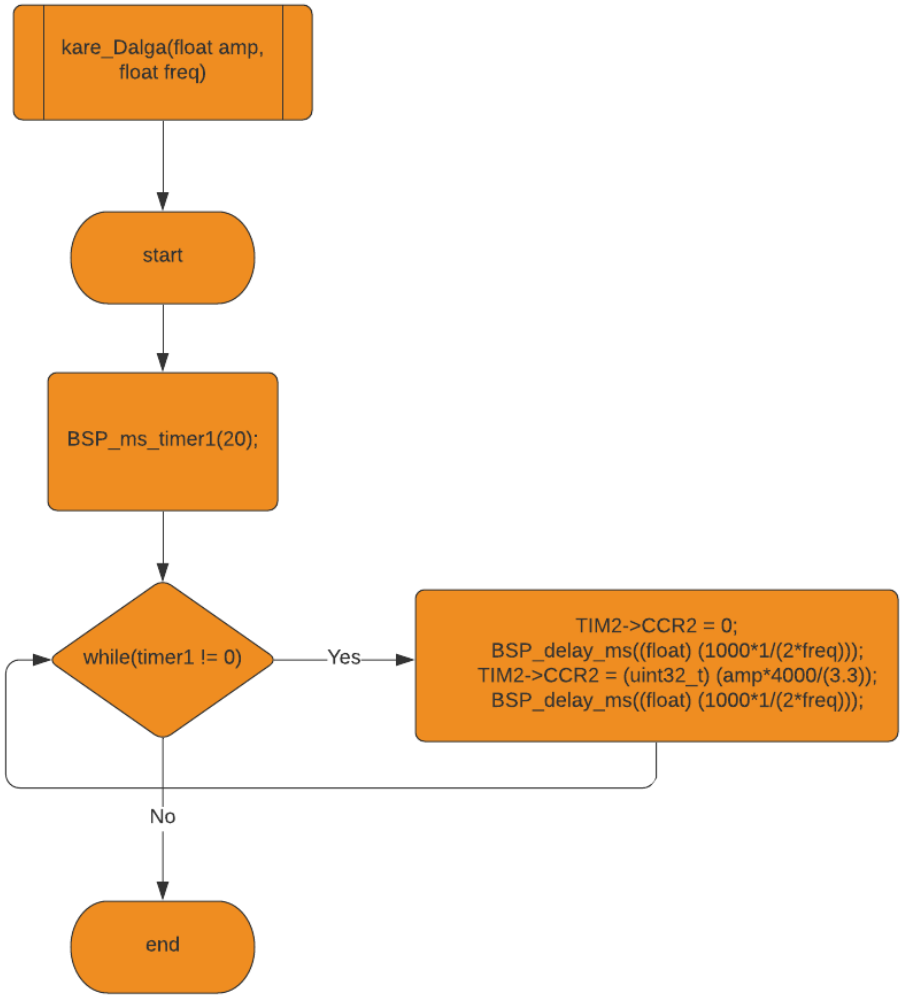

Figure 18 : Flowchart of the square wave function

Important functions from bsp.c for the project

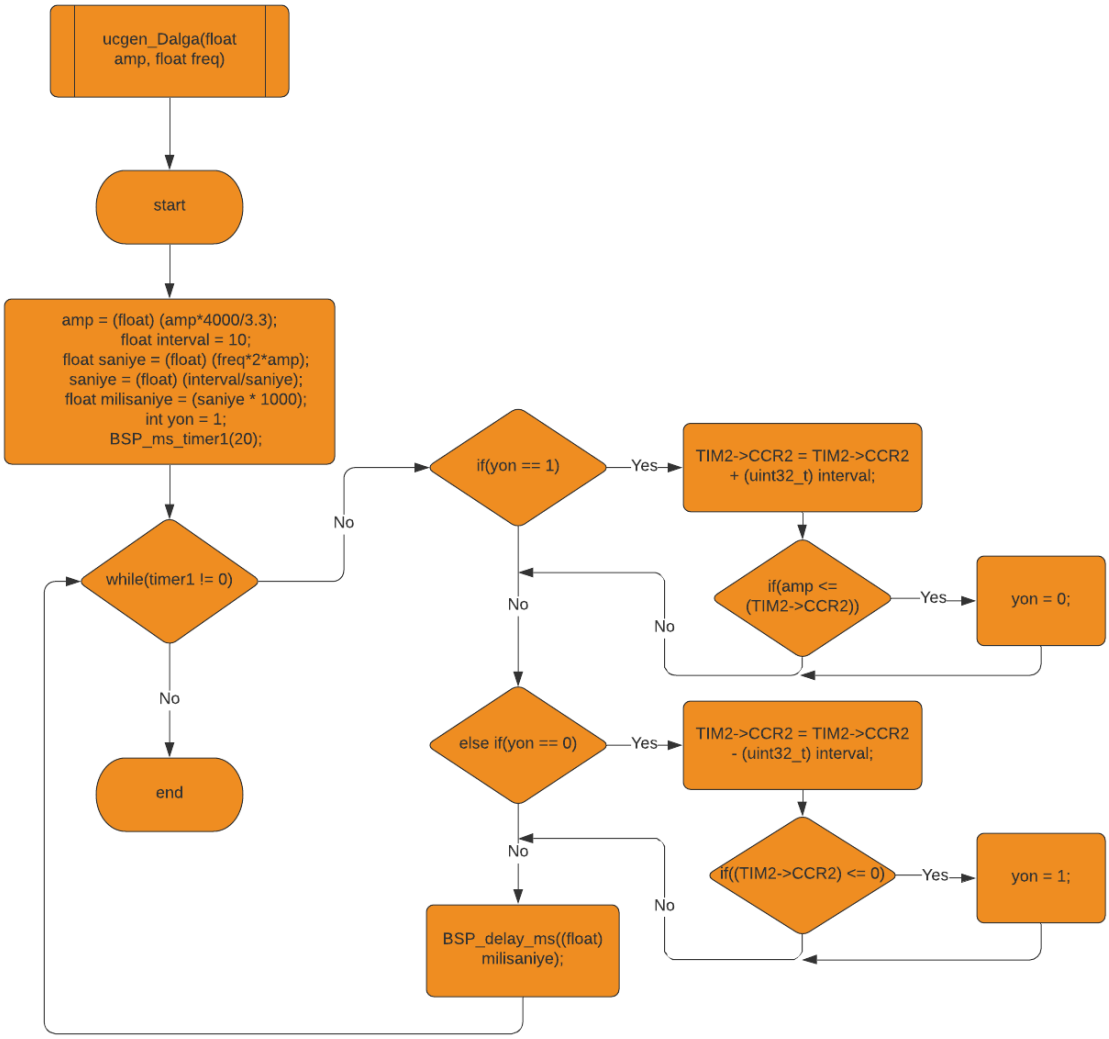

Figure 19 : Flowchart of triangular wave function

Important functions from bsp.c for the project

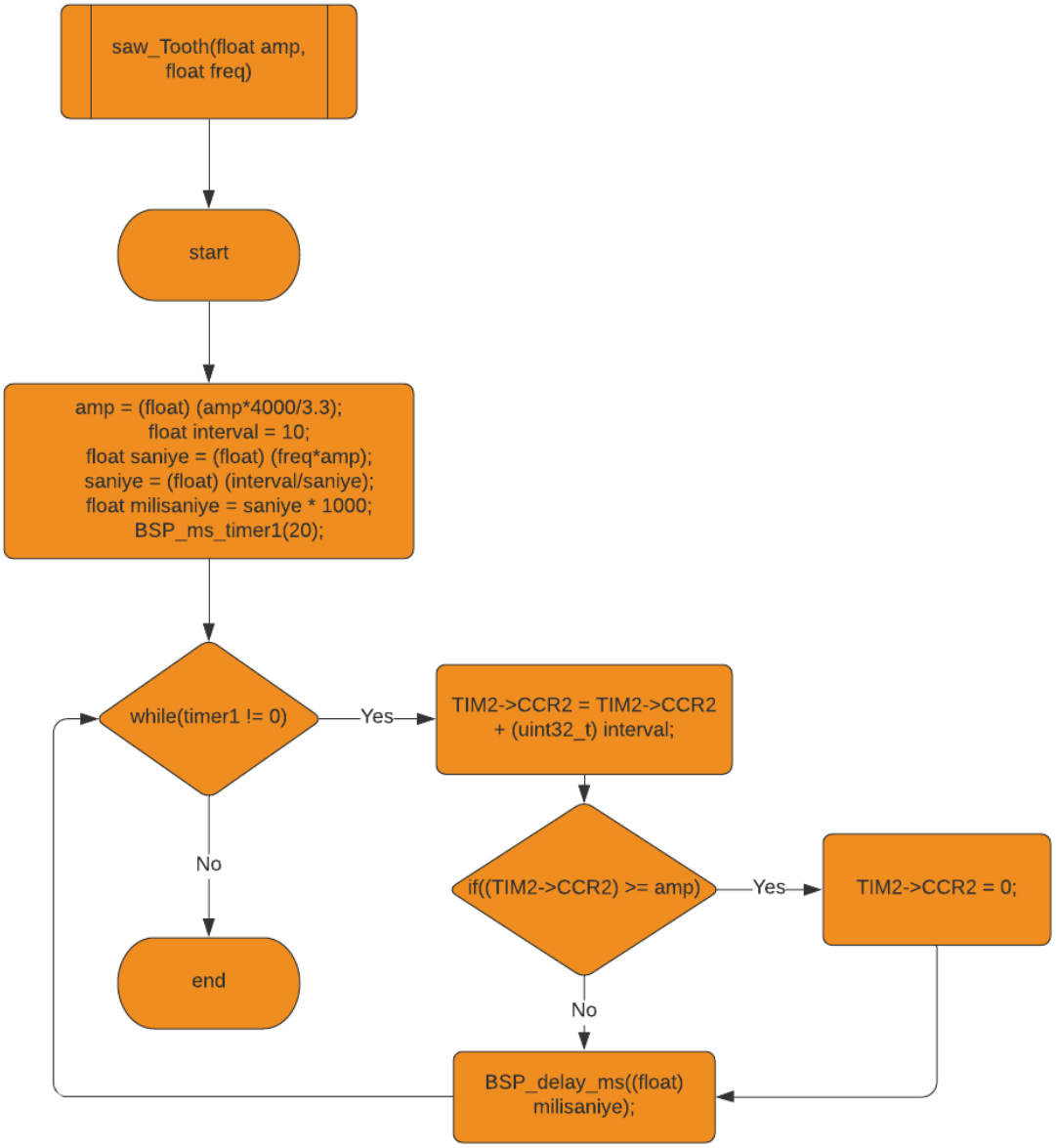

Figure 20 : Flowchart of sawTooth wave function

Important functions from bsp.c for the project

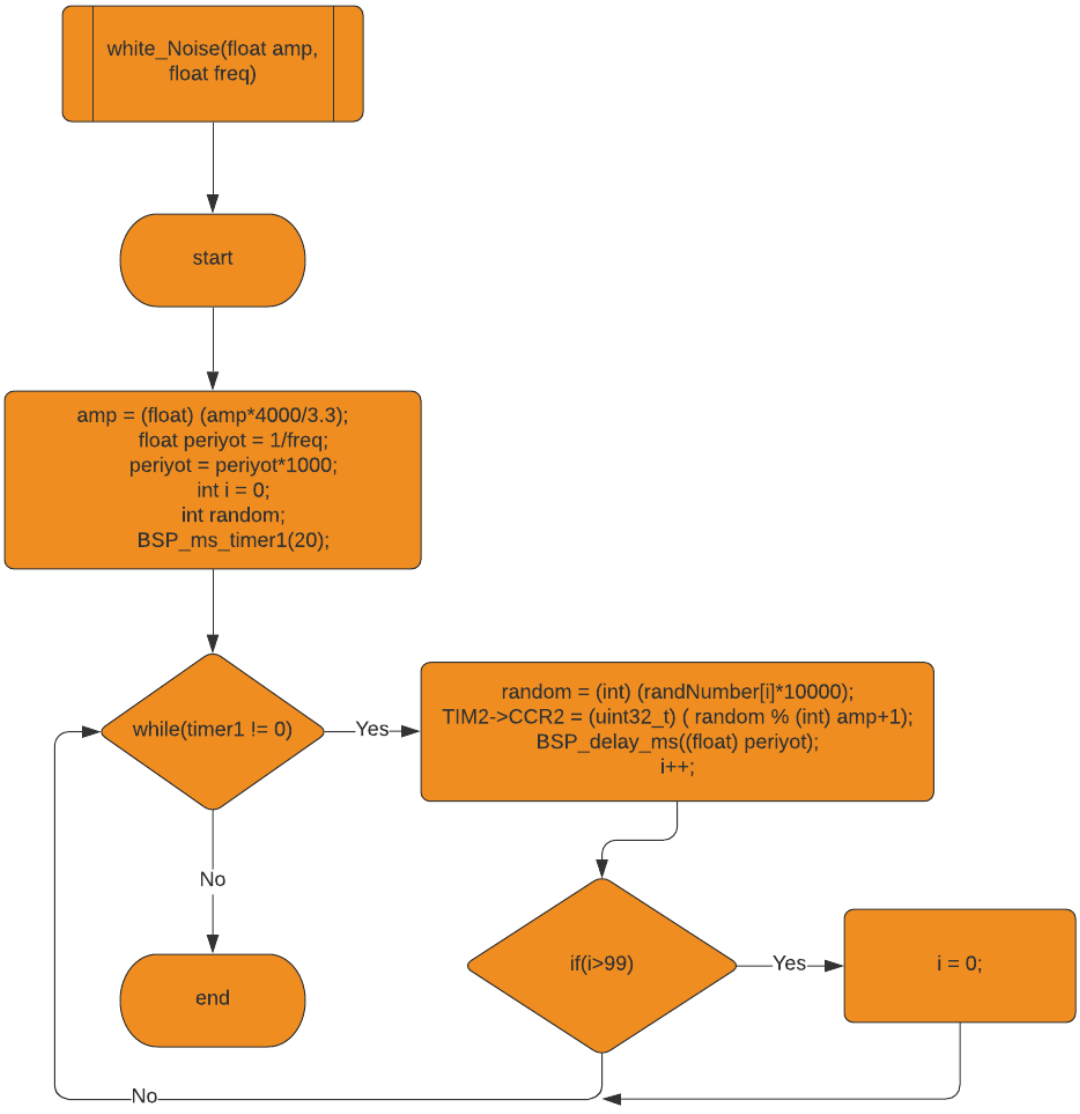

Figure 21 : Flowchart of whiteNoise wave function

Important functions from bsp.c for the project

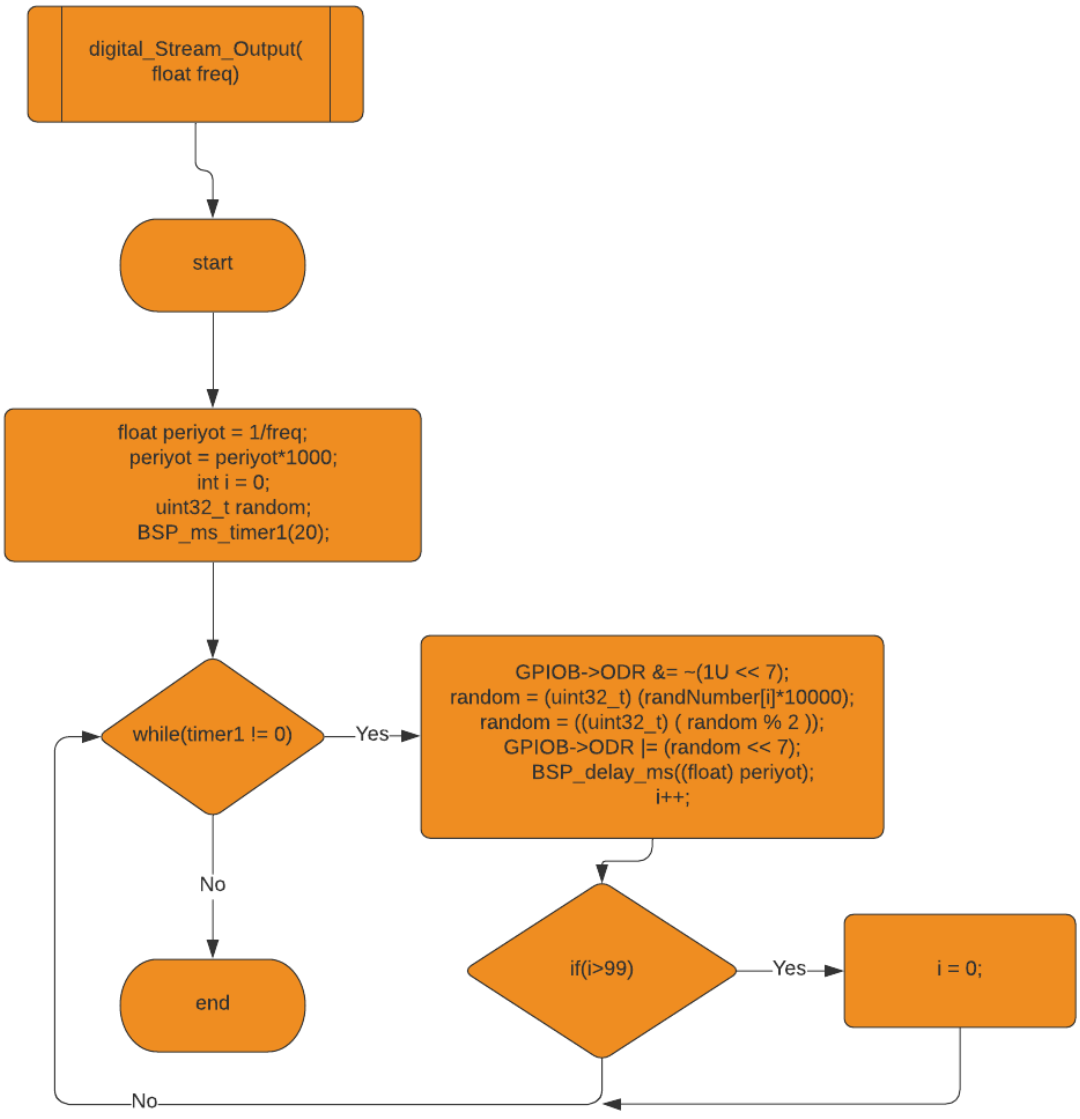

Figure 22 : Flowchart of Digital Stream Output function

Important functions from bsp.c for the project

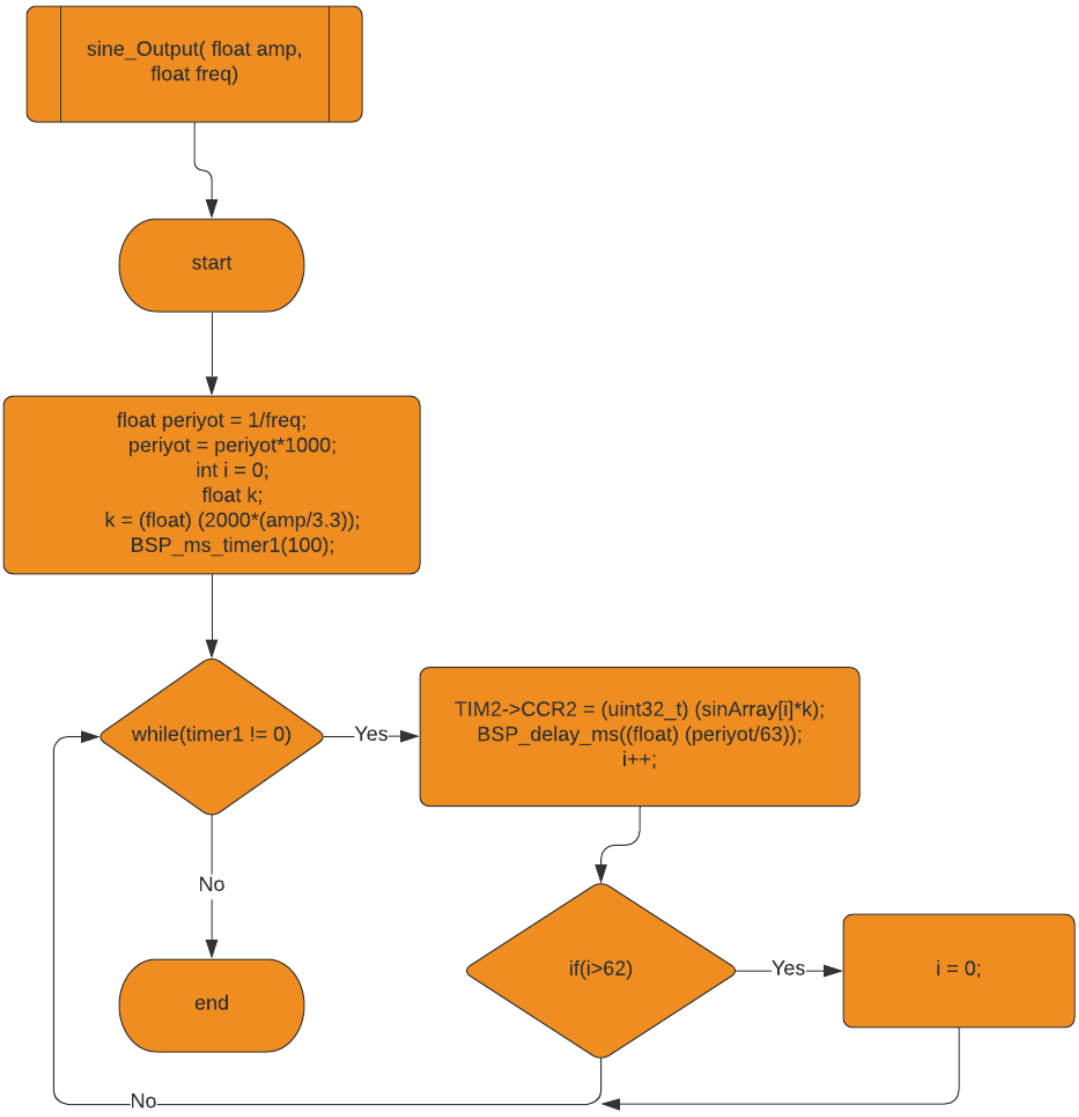

Figure 23 : Flowchart of Sinuzoidal wave function

9 BLOCK DIAGRAM

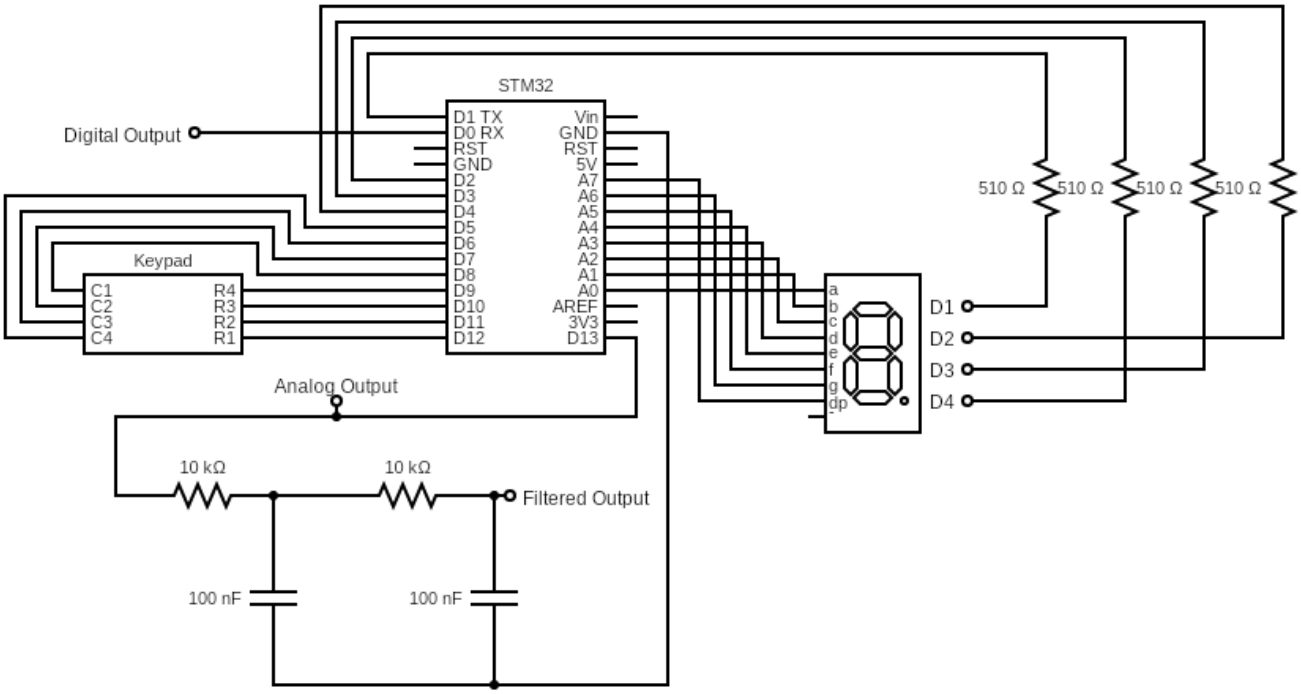

Figure 24 : Block Diagram of the circuit

10 PROJECT CONCLUSION

In this project, digital analog conversion process is learned. It is learned how the wave functions in the state library are actually studied. Stabilizing keypad and ssd has been learned.