GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC334 – MICROPROCESSORS

Project 1

HAZIRLAYANLAR
1801022035 – Ruveyda Dilara Günal
1801022071 – Alperen Arslan
1901022255 – Emirhan Köse

Requirements

● Button # will be used as an Enter key

● Button * will be used as a dot key

● Button A will be used to define the **Amplitude** of the signal. It will expect a series of numbers after it, and upon pressing the Enter key, it will apply the amplitude. E.g. A1*20# will set the amplitude to 1.20 Volts Peak-Peak.

● Button B will be used to define the **Frequency** of the signal. It will expect a series of numbers after it, and upon pressing the Enter key, it will apply the frequency. E.g. B120# will set the frequency to 120 Hz.

● There should be proper guards for voltage/amplitude inputs and other buttons (i.e. Pressing B after A should cancel the operation)

 ○ If no button is pressed for 10 seconds, timeout should happen, and input should be cancelled.

● Button C will cycle through modes.

● Button D will cycle through displaying the Mode, Amplitude and Frequency of the currently active signal.

● *Sine, square, triangle, sawtooth,* and *white noise* modes should accept both amplitude and frequency information.

● *Digital stream output* mode should randomly generate 0 and 1s at the given frequency. The amplitude should not affect the amplitude of this operation. **This should use a different output pin than the one used for other modes.**

● All operations should be shown on the SSDs. ○ By default, the SSD should display the mode. Sine, square, triangle, sawtooth (ramp), white noise and random digital stream modes. (try to spell each uniquely. ie. for sine, you can write SInE, for triangle, you can write trI, etc.)

 ○ Pressing button A should empty the SSD and each digit press should push the digit from left and shift the others to the left. . (dot) should be displayed as one digit character of your choosing)

 ○ Pressing Enter should clear the SSD and should go back to displaying the mode.

● There should be no bouncing on the buttons.

● There should be no brightness difference between Seven Segments.

● There should be no flickering on the SSDs.

● You should calculate and figure out the maximum possible frequencies that you can run.

● Output signals should be clean, so design proper filters based on the frequency range that you support.

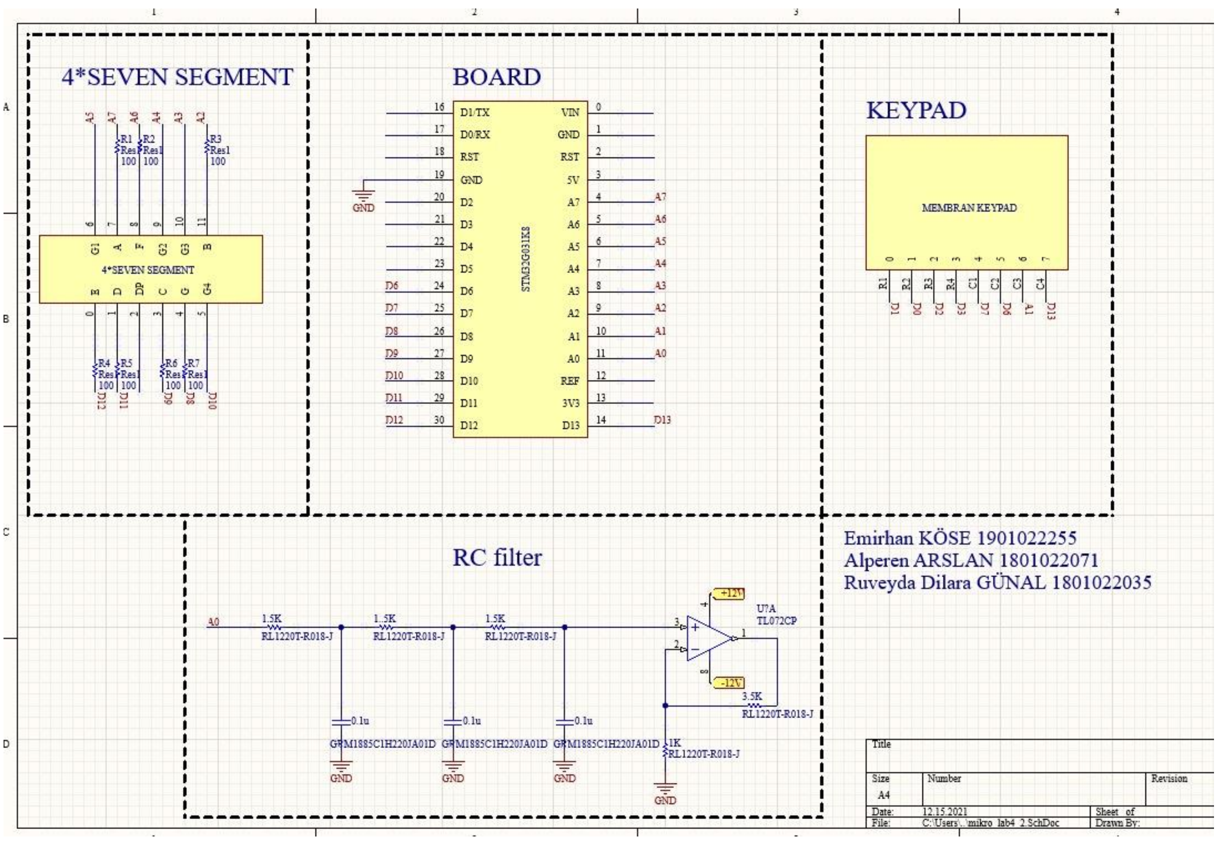

Figure 1: Block Diagram

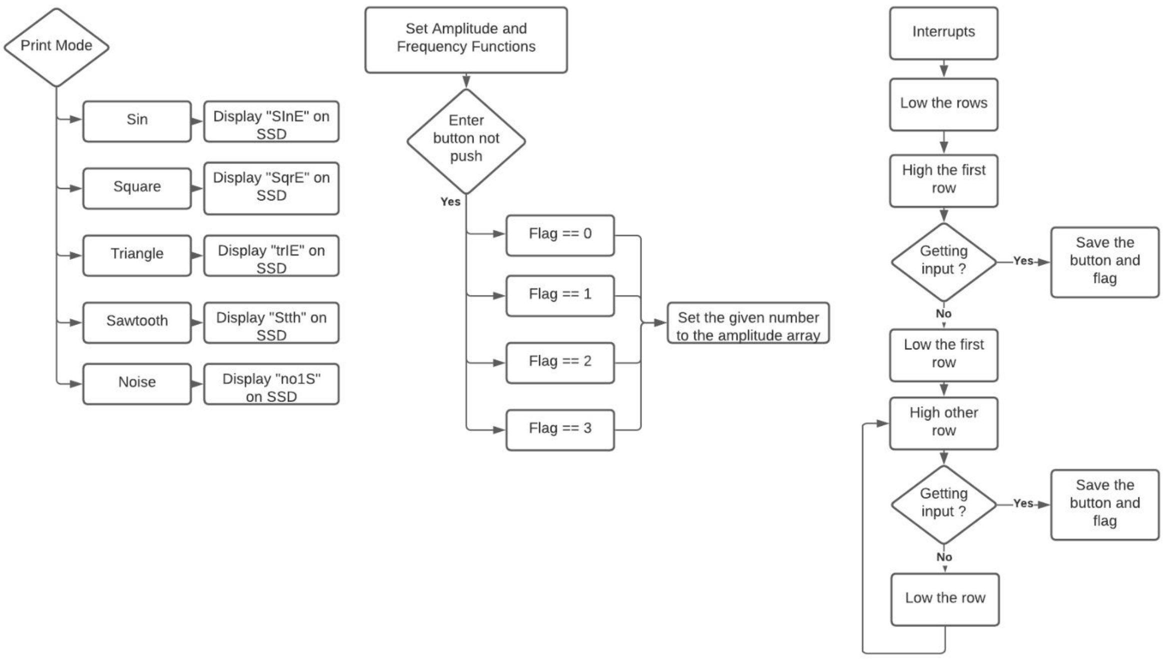

Figure 2: Flowchart

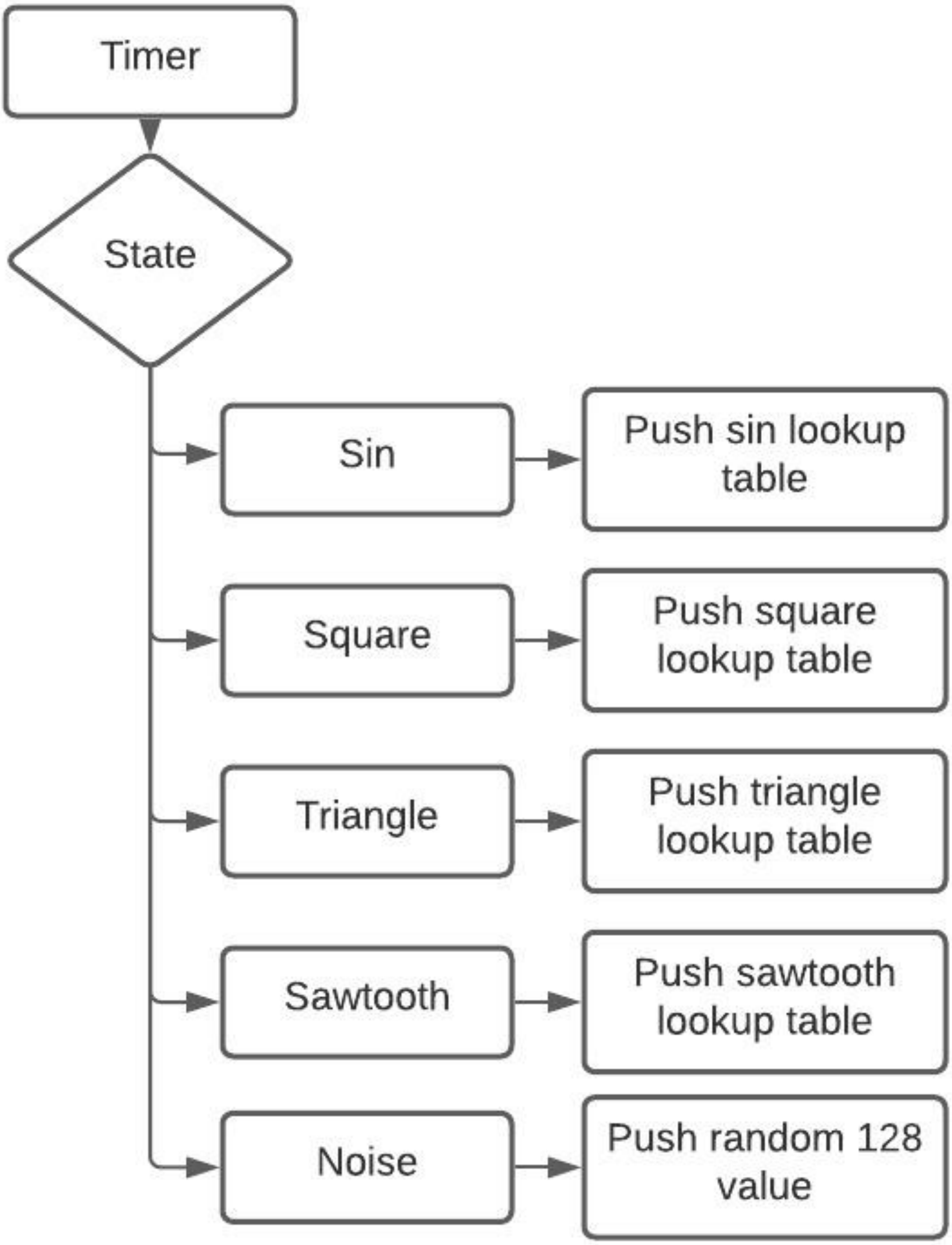

Figure 3: Flowchart

Bsp.h

```
/*

 * BSP.h

*  Author: Emirhan Köse, Alperen Arslan, Ruveyda Dilara Günal

 */

#include "stm32g0xx.h"

#include "system_stm32g0xx.h"

#ifndef BSP_H_

#define BSP_H_

#define LEDDELAY 1600000

#define KEY1   0x45670123

#define KEY2   0xCDEF89AB



#if !defined  (HSE_VALUE)

#define HSE_VALUE    (8000000UL)    /*!< Value of the External oscillator in Hz */

#endif /* HSE_VALUE */



#if !defined  (HSI_VALUE)

  #define HSI_VALUE  (16000000UL)   /*!< Value of the Internal oscillator in Hz*/

#endif /* HSI_VALUE */



#define PLL_M   1U

#define PLL_N   8U

#define PLL_P   7U

#define PLL_Q   2U
```

```c
#define PLL_L   2U


extern uint32_t SystemCoreClock;


/*COOL FUNCTIONS*/
void openClock(char port);
void setMode(char port, uint32_t num,char IO);


/*onboard led functions*/
void configureOnboardLed();
void toggleOnboardLed();
void turnOnOnboardLed();
void turnOffOnboardLed();


/*onboard button functions*/
void unlockFlash();
void lockFlash();
void configureOnboardButton();
int readOnboardButton();


/*clock configure functions*/
void set_sysclk_to_hse();
void set_sysclk_to_hsi();
void set_sysclk_to_84();


/*INTERRUPTS*/
```

```
void EXT0_1_IRQHandler();

void EXT2_3_IRQHandler();

void EXT4_15_IRQHandler();

void configure_A0_int();


/*SYSTICK functions*/

void SysTick_Handler();

void init_systick(uint32_t s);


void delay(volatile uint32_t s);

void delay_ms(uint32_t s);


/*SSD functions*/

void print_digit(volatile uint32_t dig);

void ssd_output(uint32_t x);


/*keypad functions*/

void config_keypad_pins();

void config_keypad_IRQs();

void clear_rows_keypad();

void set_rows_keypad();

#endif /* BSP_H_ */
```

Bsp.c

```
/*

 * BSP.c
```

```
 *      Author: Emirhan Köse, Alperen Arslan , Ruveyda Dilara Günal
 */
#include "BSP.h"
#include "stm32g0xx.h"
#include "system_stm32g0xx.h"


static uint32_t tDelay;
extern uint32_t SystemCoreClock;


//extern volatile uint8_t enter_flag = 0; // check if enter (#
button) is pressed


/*delay function*/
void delay(volatile uint32_t s){
	for(; s>0; s--);
}
/*COOL FUNCTIONS*/
void openClock(char port){


	switch(port){
	case 'A':
		RCC-> IOPENR |= (1U << 0);
		break;
	case 'B':
		RCC->IOPENR |= (1U << 1);
		break;
	case 'C':
```

```
            RCC->IOPENR |= (1U << 2);
            break;
      case 'D':
            RCC->IOPENR |= (1U << 3);
            break;
      case 'F':
            RCC->IOPENR |= (1U << 5);
            break;
      }
}
void setMode(char port, uint32_t num, char IO){
      switch(port){
      case 'A':
            if(num == 2 || num == 3){//dont touch PA2 and PA3 ports
even user want to change them
            break;
            }
            GPIOA-> MODER &= ~(3U << num*2); // set 0 both bytes
(input mode)
            if(IO == 'O'){//output mode
                  GPIOA-> MODER |= (1U << num*2);
            }
            else if(IO == 'I'){
                  //do nothing
            }
            else if(IO == 'A'){//analog input mode
                  GPIOA-> MODER |= (3U << num*2);
```

```
        }
        else if(IO == 'F'){//alternate function mode
            GPIOA -> MODER |= (2U << (num*2));
        }
        break;
    case 'B':
        GPIOB-> MODER &= ~(3U << num*2); // set 0 both bytes
(input mode)
        if(IO == 'O'){//output mode
            GPIOB-> MODER |= (1U << num*2);
        }
        else if(IO == 'I'){
            //do nothing
        }
        else if(IO == 'A'){//analog input mode
            GPIOB-> MODER |= (3U << num*2);
        }
        else if(IO == 'F'){//alternate function mode
            GPIOB -> MODER |= (2U << (num*2));
        }
        break;
    case 'C':
        GPIOC-> MODER &= ~(3U << num*2); // set 0 both bytes
(input mode)
        if(IO == 'O'){//output mode
            GPIOC-> MODER |= (1U << num*2);
        }
```

```
        else if(IO == 'I'){
              //do nothing
        }
        else if(IO == 'A'){//analog input mode
              GPIOC-> MODER |= (3U << num*2);
        }
        else if(IO == 'F'){//alternate function mode
              GPIOC -> MODER |= (2U << (num*2));
        }
        break;
   case 'D':
        GPIOD-> MODER &= ~(3U << num*2); // set 0 both bytes
(input mode)
        if(IO == 'O'){//output mode
              GPIOD-> MODER |= (1U << num*2);
        }
        else if(IO == 'I'){
              //do nothing
        }
        else if(IO == 'A'){//analog input mode
              GPIOD-> MODER |= (3U << num*2);
        }
        else if(IO == 'F'){//alternate function mode
              GPIOD -> MODER |= (2U << (num*2));
        }
        break;
   case 'F':
```

```
            GPIOF-> MODER &= ~(3U << num*2); // set 0 both bytes
(input mode)

            if(IO == 'O'){//output mode

                  GPIOF-> MODER |= (1U << num*2);

            }

            else if(IO == 'I'){

                  //do nothing

            }

            else if(IO == 'A'){//analog input mode

                  GPIOF-> MODER |= (3U << num*2);

            }

            else if(IO == 'F'){//alternate function mode

                  GPIOF -> MODER |= (2U << (num*2));

            }

            break;

      }

}

/*onboard led functions*/

void configureOnboardLed(){

      RCC->IOPENR |= (1U << 2);

      /* Setup PC6 as output */

      GPIOC->MODER &= ~(3U << 2*6);

      GPIOC->MODER |= (1U << 2*6);

}

void toggleOnboardLed(){

          /* Turn on LED */

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

```
        while(1) {
            delay(LEDDELAY);
            /* Toggle LED */
            GPIOC->ODR ^= (1U << 6);
        }
}
void turnOnOnboardLed(){
            /* Turn on LED */
            GPIOC->ODR |= (1U << 6);
}
void turnOffOnboardLed(){
                /* Turn off LED */
                        GPIOC->ODR &= ~(1U << 6);
}
/*onboard Button Functions*/
void unlockFlash() {
    if (FLASH->CR & FLASH_CR_LOCK) {
        FLASH->KEYR = KEY1;
        FLASH->KEYR = KEY2;
    }
}
void lockFlash() {
    FLASH->CR |= FLASH_CR_LOCK; // bit 31
}
void configureOnboardButton(){
     /*activate clock for the port f*/
```

```
	RCC-> IOPENR |= (1U << 5);
	/*enable change the optr by clearing the lock bit*/
	unlockFlash();
	/*change button mode reset to GPIO*/
	FLASH -> OPTR &= ~(3U << 27);
	FLASH -> OPTR |= (1U << 27);

	/*setup PF2 as input*/
	GPIOF -> MODER &= ~(3U << 2*2);
	//GPIOF->MODER |= (1U << 2*2);
	//GPIOF-> ODR |= (1U << 2);
}
int readOnboardButton(){   //torigari cindari
	if(((GPIOF -> IDR)) & 4U){
		return 1;//if the onboard led is pressed, return 1
	}
	return 0;
}
/*processor clock functions*/

void set_sysclk_to_hse(){

	SystemInit();
	//enable HSE
	RCC->CR |= (1 << 16);
	//wait till HSE is ready
```

```
	while(!(RCC->CR & (1 << 17)));


	/*configure flash*/

	FLASH->ACR = (1 << 8) | (1 << 9) | (1 << 10 ) | (0 << 0);

	//select HSE as system clock

	RCC->CFGR &= ~(3U << 0);

	RCC->CFGR |=  (1 << 0);

	//wait till the PPL used as system clock

	while (!(RCC->CFGR & (1 << 2)));

    SystemCoreClock = HSE_VALUE;

}

void set_sysclk_to_hsi(){


	/* Reset goes to HSI by default */

	   SystemInit();


	   /* Configure Flash

	    * prefetch enable (ACR:bit 8)

	    * instruction cache enable (ACR:bit 9)

	    * data cache enable (ACR:bit 10)

	    * set latency to 0 wait states (ARC:bits 2:0)

	    *   see Table 10 on page 80 in RM0090

	    */

	   FLASH->ACR = (1 << 8) | (1 << 9) | (1 << 10 ) | (0 << 0);

	   SystemCoreClock = HSI_VALUE;

}
```

```
void set_sysclk_to_64(){   //torigari cindari???


     SystemInit();

         #undef PLL_P

         uint32_t PLL_P = 4;


         /* Enable HSE (CR: bit 16) */

         RCC->CR |= (1 << 16);

         /* Wait till HSE is ready (CR: bit 17) */

         while(!(RCC->CR & (1 << 17)));


         /* set voltage scale to 1 for max frequency */

         /* first enable power interface clock (APB1ENR:bit 28) */

         RCC->APBENR1 |= (1 << 28);


         /* then set voltage scale to 1 for max frequency
(PWR_CR:bit 14)

          * (0) scale 2 for fCLK <= 144 Mhz

          * (1) scale 1 for 144 Mhz < fCLK <= 168 Mhz

          */

         PWR->CR1 |= (1 << 14);


         /* set AHB prescaler to /1 (CFGR:bits 7:4) */

         RCC->CFGR |= (0 << 4);

         /* set ABP low speed prescaler to /4 (APB1) (CFGR:bits
12:10) */

         RCC->CFGR |= (5 << 10);
```

```
        /* set ABP high speed prescaper to /2 (ABP2) (CFGR:bits
15:13) */

        RCC->CFGR |= (4 << 13);


        /* Set M, N, P and Q PLL dividers

         * PLLCFGR: bits 5:0 (M), 14:6 (N), 17:16 (P), 27:24 (Q)

         * Set PLL source to HSE, PLLCFGR: bit 22, 1:HSE, 0:HSI

         */

        RCC->PLLCFGR = PLL_M | (PLL_N << 6) | (((PLL_P >> 1) -1)
<< 16) |

                       (PLL_Q << 24) | (1 << 22);

        /* Enable the main PLL (CR: bit 24) */

        RCC->CR |= (1 << 24);

        /* Wait till the main PLL is ready (CR: bit 25) */

        while(!(RCC->CR & (1 << 25)));

        /* Configure Flash

         * prefetch enable (ACR:bit 8)

         * instruction cache enable (ACR:bit 9)

         * data cache enable (ACR:bit 10)

         * set latency to 2 wait states (ARC:bits 2:0)

         *   see Table 10 on page 80 in RM0090

         */

        FLASH->ACR = (1 << 8) | (1 << 9) | (1 << 10 ) | (2 << 0);


        /* Select the main PLL as system clock source, (CFGR:bits
1:0)

         * 0b00 - HSI
```

```
         * 0b01 - HSE
         * 0b10 - PLL
         */
        RCC->CFGR &= ~(3U << 0);
        RCC->CFGR |=  (2  << 0);
        /* Wait till the main PLL is used as system clock source
(CFGR:bits 3:2) */
        while (!(RCC->CFGR & (2 << 2)));


        SystemCoreClock = 64000000;
}
/*
void EXTI4_15IRQHandler(void){
}
*/
void configure_A0_int(){/*
    RCC-> APBENR2 |= (1U << 0); //enable SYSCFG clock
    EXTI-> EXTICR[0] |= (0U << 8*0); //chose port A (0. port) and
0th pin  (8*0)


     EXTI->RTSR1 |= (1U << 0);//chose falling edge trigger at A0
(0th pin, so shift 0 bits to the left)
        EXTI->IMR1 |= (1U << 0);  // Mask pin 0
          NVIC_SetPriority(EXTI0_1_IRQn,1);
          NVIC_EnableIRQ(EXTI0_1_IRQn);
*/
}
```

```
/*SYSTICK functions*/

void SysTick_Handler(void)

{

    if (tDelay != 0)

    {

        tDelay--;

    }

}

void init_systick(uint32_t s){

    // Clear CTRL register

    SysTick->CTRL = 0x00000;

    // Main clock source is running with HSI by default which is at
8 Mhz.

    // SysTick clock source can be set with CTRL register (Bit 2)

    // 0: Processor clock/8 (AHB/8)

    // 1: Processor clock (AHB)

    SysTick->CTRL |= (1 << 2);

    // Enable callback (bit 1)

    SysTick->CTRL |= (1 << 1);

    // Load the value

    SysTick->LOAD = (uint32_t)(s-1);

    // Set the current value to 0

    SysTick->VAL = 0;

    // Enable SysTick (bit 0)

    SysTick->CTRL |= (1 << 0);

}

void delay_ms(uint32_t s)
```

```
{
    tDelay = s;
    while(tDelay != 0);
}
/*SSD functions*/
void print_digit(volatile uint32_t dig)//SSD digit print function
{
      switch (dig){
      case 0:
            GPIOB->ODR |= (1U << 4);   //set  d12
            GPIOB->ODR |= (1U << 5);   //set  d11
            GPIOA->ODR |= (1U << 8);   //set  d9
            GPIOA->ODR |= (1U << 4);   //set  a2
            GPIOA->ODR |= (1U << 6);   //set  a6
            GPIOA->ODR |= (1U << 7);   //set  a7
            GPIOB->ODR &= ~(1U << 8);  //set  d8
            break;
      case 1:
            GPIOA->ODR |= (1U << 8);   //set  d9
            GPIOA->ODR |= (1U << 4);   //set  a2
            GPIOB->ODR &= ~(1U << 4);  //set  d12
            GPIOB->ODR &= ~(1U << 5);  //set  d11
            GPIOB->ODR &= ~(1U << 8);  //set  d8
            GPIOA->ODR &= ~(1U << 6);  //set  a6
            GPIOA->ODR &= ~(1U << 7);  //set  a7
            break;
```

```
case 2:
    GPIOB->ODR |= (1U << 4);   //set  d12
    GPIOB->ODR |= (1U << 5);   //set  d11
    GPIOB->ODR |= (1U << 8);   //set  d8
    GPIOA->ODR |= (1U << 4);   //set  a2
    GPIOA->ODR |= (1U << 7);   //set  a7
    GPIOA->ODR &= ~(1U << 8);  //set  d9
    GPIOA->ODR &= ~(1U << 6);  //set  a6
    break;
case 3:
    GPIOB->ODR |= (1U << 5);   //set  d11
    GPIOA->ODR |= (1U << 8);   //set  d9
    GPIOB->ODR |= (1U << 8);   //set  d8
    GPIOA->ODR |= (1U << 4);   //set  a2
    GPIOA->ODR |= (1U << 7);   //set  a7
    GPIOB->ODR &= ~(1U << 4);  //reset  d12
    GPIOA->ODR &= ~(1U << 6);  //reset  a6
    break;
case 4:
    GPIOA->ODR |= (1U << 8);   //set  d9
    GPIOB->ODR |= (1U << 8);   //set  d8
    GPIOA->ODR |= (1U << 4);   //set  a2
    GPIOA->ODR |= (1U << 6);   //set  a6
    GPIOB->ODR &= ~(1U << 4);  //reset  d12
    GPIOB->ODR &= ~(1U << 5);  //reset  d11
    GPIOA->ODR &= ~(1U << 7);  //reset  a7
```

```
        break;
    case 5:
        GPIOB->ODR |= (1U << 5);   //set  d11
        GPIOA->ODR |= (1U << 8);   //set  d9
        GPIOB->ODR |= (1U << 8);   //set  d8
        GPIOA->ODR |= (1U << 6);   //set  a6
        GPIOA->ODR |= (1U << 7);   //set  a7
        GPIOB->ODR &= ~(1U << 4);  //reset  d12
        GPIOA->ODR &= ~(1U << 4);  //reset  a2
        break;
    case 6:
        GPIOB->ODR |= (1U << 4);   //set  d12
        GPIOB->ODR |= (1U << 5);   //set  d11
        GPIOA->ODR |= (1U << 8);   //set  d9
        GPIOB->ODR |= (1U << 8);   //set  d8
        GPIOA->ODR |= (1U << 6);   //set  a6
        GPIOA->ODR |= (1U << 7);   //set  a7
        GPIOA->ODR &= ~(1U << 4);  //reset  a2
        break;
    case 7:
        GPIOA->ODR |= (1U << 8);   //set  d9
        GPIOA->ODR |= (1U << 4);   //set  a2
        GPIOA->ODR |= (1U << 6);   //set  a6
        GPIOA->ODR |= (1U << 7);   //set  a7
        GPIOB->ODR &= ~(1U << 4);  //reset  d12
        GPIOB->ODR &= ~(1U << 5);  //reset  d11
```

```
        GPIOB->ODR &= ~(1U << 8);  //reset  d8
        break;
    case 8:
        GPIOB->ODR |= (1U << 4);   //set  d12
        GPIOB->ODR |= (1U << 5);   //set  d11
        GPIOA->ODR |= (1U << 8);   //set  d9
        GPIOB->ODR |= (1U << 8);   //set  d8
        GPIOA->ODR |= (1U << 4);   //set  a2
        GPIOA->ODR |= (1U << 6);   //set  a6
        GPIOA->ODR |= (1U << 7);   //set  a7
        break;
    case 9:
        GPIOB->ODR |= (1U << 5);   //set  d11
        GPIOA->ODR |= (1U << 8);   //set  d9
        GPIOB->ODR |= (1U << 8);   //set  d8
        GPIOA->ODR |= (1U << 4);   //set  a2
        GPIOA->ODR |= (1U << 6);   //set  a6
        GPIOA->ODR |= (1U << 7);   //set  a7
        GPIOB->ODR &= ~(1U << 4);  //reset  d12
        break;
    case 10: //letter for enter key (#)
        GPIOB->ODR |= (1U << 5);   //set  d11
        GPIOA->ODR |= (1U << 8);   //set  d9
        GPIOB->ODR |= (1U << 8);   //set  d8
        GPIOA->ODR |= (1U << 4);   //set  a2
        GPIOA->ODR |= (1U << 7);   //set  a
```

```
        GPIOA->ODR &= ~(1U << 6);  //set  a6
        GPIOB->ODR &= ~(1U << 4);  //reset  d12
        break;
    case 11: //letter A
        GPIOB->ODR |= (1U << 4);   //set  d12
        GPIOA->ODR |= (1U << 4);   //set  a2
        GPIOA->ODR |= (1U << 6);   //set  a6
        GPIOA->ODR |= (1U << 7);   //set  a7
        GPIOB->ODR |= (1U << 8);   //set  d8
        GPIOA->ODR |= (1U << 8);   //set  d9
        GPIOB->ODR &= ~(1U << 5);  //set  d11
        break;
    case 12://letter c
        GPIOA->ODR |= (1U << 7);   //set  a
        GPIOA->ODR &= ~(1U << 4);  //set  b
        GPIOA->ODR &= ~(1U << 8);  //set  c
        GPIOB->ODR |= (1U << 5);   //set  d
        GPIOB->ODR |= (1U << 4);   //set  e
        GPIOA->ODR |= (1U << 6);   //set  f
        GPIOB->ODR &= ~(1U << 8);  //set  g
        break;
    case 13: // letter E
            GPIOA->ODR |= (1U << 7);   //set  a
            GPIOA->ODR &= ~(1U << 4);  //set  b
            GPIOA->ODR &= ~(1U << 8);  //set  c
            GPIOB->ODR |= (1U << 5);   //set  d
```

```
            GPIOB->ODR |= (1U << 4);   //set  e
            GPIOA->ODR |= (1U << 6);   //set  f
            GPIOB->ODR |= (1U << 8);   //set  g
            break;
    case 14: //letter n
            GPIOA->ODR &= ~(1U << 7);  //set  a
            GPIOA->ODR &= ~(1U << 4);  //set  b
            GPIOA->ODR |= (1U << 8);   //set  c
            GPIOB->ODR &= ~(1U << 5);  //set  d
            GPIOB->ODR |= (1U << 4);   //set  e
            GPIOA->ODR &= ~(1U << 6);  //set  f
            GPIOB->ODR |= (1U << 8);   //set  g
            break;
    case 15: //letter t
            GPIOA->ODR &= ~(1U << 7);  //set  a
            GPIOA->ODR &= ~(1U << 4);  //set  b
            GPIOA->ODR &= ~(1U << 8);  //set  c
            GPIOB->ODR |= (1U << 5);   //set  d
            GPIOB->ODR |= (1U << 4);   //set  e
            GPIOA->ODR |= (1U << 6);   //set  f
            GPIOB->ODR |= (1U << 8);   //set  g
            break;
    case 16: //letter r
            GPIOA->ODR &= ~(1U << 7);  //set  a
            GPIOA->ODR &= ~(1U << 4);  //set  b
            GPIOA->ODR &= ~(1U << 8);  //set  c
```

```
            GPIOB->ODR &= ~(1U << 5);  //set  d
            GPIOB->ODR |= (1U << 4);   //set  e
            GPIOA->ODR &= ~(1U << 6);  //set  f
            GPIOB->ODR |= (1U << 8);   //set  g
            break;
    case 17: //letter H
            GPIOA->ODR &= ~(1U << 7);  //set  a
            GPIOA->ODR |= (1U << 4);   //set  b
            GPIOA->ODR |= (1U << 8);   //set  c
            GPIOB->ODR &= ~(1U << 5);  //set  d
            GPIOB->ODR |= (1U << 4);   //set  e
            GPIOA->ODR |= (1U << 6);   //set  f
            GPIOB->ODR |= (1U << 8);   //set  g
            break;
    case 18: // letter q
            GPIOA->ODR |= (1U << 7);   //set  a
            GPIOA->ODR |= (1U << 4);   //set  b
            GPIOA->ODR |= (1U << 8);   //set  c
            GPIOB->ODR &= ~(1U << 5);  //set  d
            GPIOB->ODR &= ~(1U << 4);  //set  e
            GPIOA->ODR |= (1U << 6);   //set  f
            GPIOB->ODR |= (1U << 8);   //set  g
            break;
    case 19://dot symbol
            GPIOA->ODR &= ~(1U << 7);  //set  a
            GPIOA->ODR &= ~(1U << 4);  //set  b
```

```
                GPIOA->ODR &= ~(1U << 8);  //set  c
                GPIOB->ODR |= (1U << 5);   //set  d
                GPIOB->ODR &= ~(1U << 4);  //set  e
                GPIOA->ODR &= ~(1U << 6);  //set  f
                GPIOB->ODR &= ~(1U << 8);  //set  g
                break;
        case 20: // letter o
                GPIOA->ODR &= ~(1U << 7);  //set  a
                GPIOA->ODR &= ~(1U << 4);  //set  b
                GPIOA->ODR |= (1U << 8);   //set  c
                GPIOB->ODR |= (1U << 5);   //set  d
                GPIOB->ODR |= (1U << 4);   //set  e
                GPIOA->ODR &= ~(1U << 6);  //set  f
                GPIOB->ODR |= (1U << 8);   //set  g
        }
}
void ssd_output(uint32_t x)//special function to print 4 digit
numbers
{
            uint32_t dig1;
            uint32_t dig2;
            uint32_t dig3;
            uint32_t dig4;
            dig4 = x % 10U;
            x = x /10U;
            dig3 = x % 10U;
            x = x /10U;
```

```
      dig2 = x % 10U;
      x = x /10U;
      dig1 = x % 10U;
  //uint8_t flag = 0;
      int c = 160;
      while(c>0){

           print_digit(dig1);
           GPIOA->ODR &= ~(1U << 11); //reset  a5
           delay_ms(2);
           GPIOA->ODR |= (1U << 11);  //set  a5

           print_digit(dig2);
           GPIOA->ODR &= ~(1U << 12); //reset  a4
           delay_ms(2);
           GPIOA->ODR |= (1U << 12);  //set  a4

           print_digit(dig3);
           GPIOA->ODR &= ~(1U << 5);  //reset  a3
           delay_ms(2);
           GPIOA->ODR |= (1U << 5);   //set  a3

           print_digit(dig4);
           GPIOB->ODR &= ~(1U << 9);  //reset  d10
           delay_ms(2);
           GPIOB->ODR |= (1U << 9);   //set  d10
```

```
            c--;
            }
      }
/*keypad functions*/
void config_keypad_pins(){
      //assumed that clocks are already opened
      //rows
            setMode('B',6,'O');  //D1 -> row1
            setMode('B',7,'O');  //D0 -> row2
            setMode('A',15,'O'); //D2 -> row3
            setMode('B',1,'O');  //D3 -> row4

            //columns
            setMode('B',2,'I'); //D7 -> column1
            setMode('B',0,'I'); //D6 -> column2
            setMode('A',1,'I'); //A1 -> column3
            setMode('B',3,'I'); //D13 -> column4

            //set input pins pulldown mode for stability
            GPIOB->PUPDR |= (2U << 2*2);
            GPIOB->PUPDR |= (2U << 2*0);
            GPIOA->PUPDR |= (2U << 2*1);
            GPIOB->PUPDR |= (2U << 2*3);

            //set all rows high as initially
            GPIOB->ODR |= (1U << 6);
```

```
            GPIOB->ODR |= (1U << 7);
            GPIOA->ODR |= (1U << 15);
            GPIOB->ODR |= (1U << 1);
}
void config_keypad_IRQs(){
//config PA10,PA9,PB0,PA7;
      //RCC-> APBENR2 |= (1U << 0); //enable SYSCFG clock

      //PB2  0-1-2-3 mux
      EXTI->EXTICR[0] |= (1U << 8*2);
      EXTI->RTSR1 |= (1U << 2);
      EXTI->IMR1 |= (1U << 2);

      //PA1  0-1-2-3 mux
      EXTI->EXTICR[0] |= (0U << 8*1);
      EXTI->RTSR1 |= (1U << 1);
      EXTI->IMR1 |= (1U << 1);

      //PA0  0-1-2-3 mux
      EXTI->EXTICR[0] |= (1U << 8*0);
      EXTI->RTSR1 |= (1U << 0);
      EXTI->IMR1 |= (1U << 0);

      //PB3 0-1-2-3 mux
      EXTI->EXTICR[0] |= (1U << 8*3);
      EXTI->RTSR1 |= (1U << 3);
```

```
      EXTI->IMR1 |= (1U << 3);


    NVIC_SetPriority(EXTI0_1_IRQn,2);

    NVIC_EnableIRQ(EXTI0_1_IRQn);

      NVIC_SetPriority(EXTI2_3_IRQn,1);

      NVIC_EnableIRQ(EXTI2_3_IRQn);

    /*

    NVIC_SetPriority(EXTI4_15_IRQn,2);

    NVIC_EnableIRQ(EXTI4_15_IRQn);

*/

}


void clear_rows_keypad(){

      //set all rows low

      GPIOB->ODR &= ~(1U << 6);

      GPIOB->ODR &= ~(1U << 7);

      GPIOA->ODR &= ~(1U << 15);

      GPIOB->ODR &= ~(1U << 1);

}

void set_rows_keypad(){

      //set all rows high as initially

      GPIOB->ODR |= (1U << 6);

      GPIOB->ODR |= (1U << 7);

      GPIOA->ODR |= (1U << 15);

      GPIOB->ODR |= (1U << 1);

}
```

Main.c

```
/*
 *Author: Emirhan Köse, Alperen Arslan, Ruveyda Dilara Günal
 */
#include "stm32g0xx.h"
#include "BSP.h"
#include "system_stm32g0xx.h"
#include <stdlib.h> //for rand function
#define SYSTEM_CLK  16000000
volatile uint8_t num; //1 digit number comes from keypad
volatile uint8_t print_flag = 0; // flag for printnig the mode
volatile uint8_t amplitude_flag = 0; //flag for printing the amplitude
volatile uint8_t frequency_flag = 0;//flag for printing the frequency
volatile uint8_t enter_flag  = 0;
volatile uint8_t dot_flag = 0;
volatile uint8_t dot_print_flag = 0;
volatile uint8_t dot_index = 5;//assign is as invalid digit as initial
 uint8_t i = 0; //variable for check number of button presses before enter key
 volatile uint32_t tim_counter = 0;
volatile uint8_t print_counter = 0;
enum mode{sin,square,triangle,sawtooth,noise};


const uint32_t lookup_table[4][128] = {
          {
```

```
            //sin wave
            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
},
{
            //square
            1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023,
1023, 1023, 1023, 1023, 1023, 1023,
            1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023,
1023, 1023, 1023, 1023, 1023, 1023,
            1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023,
1023, 1023, 1023, 1023, 1023, 1023,
            1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023,
1023, 1023, 1023, 1023, 1023, 1023,
            1023, 1023, 1023, 1023, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
		0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
		0, 0, 0, 0, 0, 0, 0, 1023
	},
	{
		//triangle
		0, 16, 32, 48, 64, 81, 97, 113, 129, 145, 161, 177, 193,
209, 226,
		242, 258, 274, 290, 306, 322, 338, 354, 371, 387, 403,
419, 435, 451, 467,
		483, 499, 516, 532, 548, 564, 580, 596, 612, 628, 644,
661, 677, 693, 709,
		725, 741, 757, 773, 789, 806, 822, 838, 854, 870, 886,
902, 918, 934, 951,
		967, 983, 999, 1015, 1015, 999, 983, 967, 951, 934, 918,
902, 886, 870, 854,
		838, 822, 806, 789, 773, 757, 741, 725, 709, 693, 677,
661, 644, 628, 612,
		596, 580, 564, 548, 532, 516, 499, 483, 467, 451, 435,
419, 403, 387, 371,
		354, 338, 322, 306, 290, 274, 258, 242, 226, 209, 193,
177, 161, 145, 129,
		113, 97, 81, 64, 48, 32, 16, 0
},
{
		//sawtooth
		0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 81, 89, 97, 105,
113,
		121, 129, 137, 145, 153, 161, 169, 177, 185, 193, 201,
209, 217, 226, 234,
```

```
           242, 250, 258, 266, 274, 282, 290, 298, 306, 314, 322,
330, 338, 346, 354,
           362, 371, 379, 387, 395, 403, 411, 419, 427, 435, 443,
451, 459, 467, 475,
           483, 491, 499, 507, 516, 524, 532, 540, 548, 556, 564,
572, 580, 588, 596,
           604, 612, 620, 628, 636, 644, 652, 661, 669, 677, 685,
693, 701, 709, 717,
           725, 733, 741, 749, 757, 765, 773, 781, 789, 797, 806,
814, 822, 830, 838,
           846, 854, 862, 870, 878, 886, 894, 902, 910, 918, 926,
934, 942, 951, 959,
           967, 975, 983, 991, 999, 1007, 1015, 0
}
};
struct wave{
     float amp;
     uint8_t amp_dig[5];
     uint32_t freq;
     uint8_t freq_dig[5];
     enum mode state;
};
volatile struct wave wave;
volatile enum mode state = noise;
void reset() //function for reset the ssd
{
     GPIOB->ODR &= ~(1U << 4);  //reset  d12
     GPIOB->ODR &= ~(1U << 5);  //reset  d11
     GPIOA->ODR &= ~(1U << 8);  //reset  d9
```

```
	GPIOB->ODR &= ~(1U << 8);  //reset  d8

	GPIOA->ODR &= ~(1U << 4);  //reset  a2

	GPIOA->ODR &= ~(1U << 6);  //reset  a6

	GPIOA->ODR &= ~(1U << 7);  //reset  a7

}

void print_mode()// function for printing the modes
(sine,triangular etc.)

{

	while(print_flag){

	switch(wave.state){

	case sin:

			print_digit(5);//S

			GPIOA->ODR &= ~(1U << 11); //reset  a5

			delay_ms(2);

			GPIOA->ODR |= (1U << 11);  //set  a5

			print_digit(1);//I

			GPIOA->ODR &= ~(1U << 12); //reset  a4

			delay_ms(2);

			GPIOA->ODR |= (1U << 12);  //set  a4


			print_digit(14);//n

			GPIOA->ODR &= ~(1U << 5);  //reset  a3

			delay_ms(2);

			GPIOA->ODR |= (1U << 5);   //set  a3

			print_digit(13);//E

			GPIOB->ODR &= ~(1U << 9);  //reset  d10

			delay_ms(2);
```

```
            GPIOB->ODR |= (1U << 9);   //set  d10
        break;
    case square:
            print_digit(5);//S
            GPIOA->ODR &= ~(1U << 11); //reset  a5
            delay_ms(2);
            GPIOA->ODR |= (1U << 11);  //set  a5
            print_digit(18);//q
            GPIOA->ODR &= ~(1U << 12); //reset  a4
            delay_ms(2);
            GPIOA->ODR |= (1U << 12);  //set  a4
            print_digit(16);//r
            GPIOA->ODR &= ~(1U << 5);  //reset  a3
            delay_ms(2);
            GPIOA->ODR |= (1U << 5);   //set  a3
            print_digit(13);//E
            GPIOB->ODR &= ~(1U << 9);  //reset  d10
            delay_ms(2);
            GPIOB->ODR |= (1U << 9);   //set  d10
        break;
    case triangle
            print_digit(15);//r
            GPIOA->ODR &= ~(1U << 12); //reset  a4
            delay_ms(2);
            GPIOA->ODR |= (1U << 12);  //set  a4
            print_digit(16);//I
```

```
            GPIOA->ODR &= ~(1U << 5);  //reset  a3
            delay_ms(2);
            GPIOA->ODR |= (1U << 5);   //set  a3
            print_digit(1);//E
            GPIOB->ODR &= ~(1U << 9);  //reset  d10
            delay_ms(2);
            GPIOB->ODR |= (1U << 9);   //set  d10
        break;
    case sawtooth:
            print_digit(5);//S
            GPIOA->ODR &= ~(1U << 11); //reset  a5
            delay_ms(2);
            GPIOA->ODR |= (1U << 11);  //set  a5
            print_digit(15);//t
            GPIOA->ODR &= ~(1U << 12); //reset  a4
            delay_ms(2);
            GPIOA->ODR |= (1U << 12);  //set  a4
            print_digit(15);//t
            GPIOA->ODR &= ~(1U << 5);  //reset  a3
            delay_ms(2);
            GPIOA->ODR |= (1U << 5);   //set  a3
            print_digit(17);//H
            GPIOB->ODR &= ~(1U << 9);  //reset  d10
            delay_ms(2);
            GPIOB->ODR |= (1U << 9);   //set  d10
        break;
```

```
	case noise:
			print_digit(14);//n
			GPIOA->ODR &= ~(1U << 11); //reset  a5
			delay_ms(2);
			GPIOA->ODR |= (1U << 11);  //set  a5
			print_digit(20);//o
			GPIOA->ODR &= ~(1U << 12); //reset  a4
			delay_ms(2);
			GPIOA->ODR |= (1U << 12);  //set  a4
			print_digit(1);//1
			GPIOA->ODR &= ~(1U << 5);  //reset  a3
			delay_ms(2);
			GPIOA->ODR |= (1U << 5);   //set  a3
			print_digit(5);//S
			GPIOB->ODR &= ~(1U << 9);  //reset  d10
			delay_ms(2);
			GPIOB->ODR |= (1U << 9);   //set  d10
		break;
	}
	}
}
void print_amplitude(){
	while(print_flag){
		print_digit(wave.amp_dig[4]);
		GPIOA->ODR &= ~(1U << 11); //reset  a5
		delay_ms(2);
```

```
        GPIOA->ODR |= (1U << 11);  //set  a5
        print_digit(wave.amp_dig[3]);
        GPIOA->ODR &= ~(1U << 12); //reset  a4
        delay_ms(2);
        GPIOA->ODR |= (1U << 12);  //set  a4
        print_digit(wave.amp_dig[2]);
        GPIOA->ODR &= ~(1U << 5);  //reset  a3
        delay_ms(2);
        GPIOA->ODR |= (1U << 5);   //set  a3
        print_digit(wave.amp_dig[1]);
        GPIOB->ODR &= ~(1U << 9);  //reset  d10
        delay_ms(2);
        GPIOB->ODR |= (1U << 9);   //set  d10
    }
}
void print_frequency(){
    while(print_flag){
        print_digit(wave.freq_dig[4]);
        GPIOA->ODR &= ~(1U << 11); //reset  a5
        delay_ms(2);
        GPIOA->ODR |= (1U << 11);  //set  a5
        print_digit(wave.freq_dig[3]);
        GPIOA->ODR &= ~(1U << 12); //reset  a4
        delay_ms(2);
        GPIOA->ODR |= (1U << 12);  //set  a4
        print_digit(wave.freq_dig[2]);
```

```
            GPIOA->ODR &= ~(1U << 5);  //reset  a3
            delay_ms(2);
            GPIOA->ODR |= (1U << 5);   //set  a3
            print_digit(wave.freq_dig[1]);
            GPIOB->ODR &= ~(1U << 9);  //reset  d10
            delay_ms(2);
            GPIOB->ODR |= (1U << 9);   //set  d10
      }
}
void set_amplitude(){
      for(int j = 0; j < 5; ++j){
      wave.amp_dig[j] = 0; //initialize the digits as zero
      }
      uint8_t zero_flag = 0;
      uint8_t one_flag = 0;
      uint8_t two_flag = 0;
      uint8_t three_flag = 0;
      uint8_t four_flag = 0;
      dot_flag = 0;
      wave.amp = 0; //delete the previous amplitude value
      while(!enter_flag){
            switch(i){
            case 0:
                  if( zero_flag == 0){
                        wave.amp = num;
                        zero_flag  = 1;
```

```
        //    dig[i] = num;
        }
        break;
    case 1:
        if( one_flag == 0){
            wave.amp_dig[1] = num;

            if(dot_flag == 1){
                if(dot_print_flag == 1){
                    wave.amp_dig[1] = 19;
                    dot_print_flag = 0;
                }
            }
            else{
            wave.amp = ((wave.amp*10) + num);
            }
            one_flag  = 1;
        }
        break;
    case 2:
        if( two_flag == 0){
            //shifting operation
            wave.amp_dig[2] = wave.amp_dig[1];
            wave.amp_dig[1] = num;
            if(dot_flag == 1){
                if(dot_print_flag == 1){
```

```
                        wave.amp_dig[1] = 19;
                        dot_print_flag = 0;
                    }
                }
                else{
                wave.amp = ((wave.amp*10) + num);
                }
                two_flag  = 1;
            }
            break;
        case 3:
            if( three_flag == 0){
                //shifting operation
                wave.amp_dig[3] = wave.amp_dig[2];
                wave.amp_dig[2] = wave.amp_dig[1];
                wave.amp_dig[1] = num;
                if(dot_flag == 1){
                wave.amp = wave.amp + (float)(num *
0.1);//handle the floating point number
                    if(dot_print_flag == 1){
                        wave.amp_dig[1] = 19;
                        dot_print_flag = 0;
                    }
                }
                else{
                wave.amp = ((wave.amp*10) + num);
                }
```

```
                three_flag  = 1;
            }
            break;
        case 4:
            if( four_flag == 0){
                //shifting operation
                wave.amp_dig[4] = wave.amp_dig[3];
                wave.amp_dig[3] = wave.amp_dig[2];
                wave.amp_dig[2] = wave.amp_dig[1];
                wave.amp_dig[1] = num;
                if(dot_flag == 1){
                wave.amp = wave.amp + 
(float)(num*0.01);//handle the floating point number
                    if(dot_print_flag == 1){
                        wave.amp_dig[1] = 19;
                        dot_print_flag = 0;
                    }
                }
                else{
                wave.amp = ((wave.amp*10) + num);
                }
                four_flag  = 1;
            }
            break;
        default:
            i = 0; // if i is not 0,1,2,3 or 4, assign it to 
zero
```

```
                dot_flag = 0;//reset the dot
                break;
           }
                print_digit(wave.amp_dig[4]);
                GPIOA->ODR &= ~(1U << 11); //reset  a5
                delay_ms(2);
                GPIOA->ODR |= (1U << 11);  //set  a5

                print_digit(wave.amp_dig[3]);
                GPIOA->ODR &= ~(1U << 12); //reset  a4
                delay_ms(2);
                GPIOA->ODR |= (1U << 12);  //set  a4

                print_digit(wave.amp_dig[2]);
                GPIOA->ODR &= ~(1U << 5);  //reset  a3
                delay_ms(2);
                GPIOA->ODR |= (1U << 5);   //set  a3

                print_digit(wave.amp_dig[1]);
                GPIOB->ODR &= ~(1U << 9);  //reset  d10
                delay_ms(2);
                GPIOB->ODR |= (1U << 9);   //set  d10
           }
}
void set_frequency(){
     for(int j = 0; j < 5; ++j){
```

```
	wave.freq_dig[j] = 0; //initialize the digits as zero
	}
	uint8_t zero_flag = 0;
	uint8_t one_flag = 0;
	uint8_t two_flag = 0;
	uint8_t three_flag = 0;
	uint8_t four_flag = 0;
	wave.freq = 0;

	while(!enter_flag){
		switch(i){
				case 0:
					if( zero_flag == 0){

						wave.freq = num;
						zero_flag  = 1;
					//	dig[i] = num;
					}
					break;
				case 1:
					if( one_flag == 0){

						wave.freq_dig[1] = num;

						wave.freq = ((wave.freq*10) +
num);
```

```
                        one_flag  = 1;
                    }
                    break;
                case 2:
                    if( two_flag == 0){
                        //shifting operation
                        wave.freq_dig[2] =
wave.freq_dig[1];
                        wave.freq_dig[1] = num;
                        wave.freq = ((wave.freq*10) +
num);
                        two_flag  = 1;


                    }
                    break;
                case 3:
                    if( three_flag == 0){


                        //shifting operation
                        wave.freq_dig[3] =
wave.freq_dig[2];
                        wave.freq_dig[2] =
wave.freq_dig[1];
                        wave.freq_dig[1] = num;
                        wave.freq = ((wave.freq*10) +
num);
                        three_flag  = 1;
                    }
```

```
                        break;
                case 4:
                        if( four_flag == 0){
                                //shifting operation
                                wave.freq_dig[4] =
wave.freq_dig[3];
                                wave.freq_dig[3] =
wave.freq_dig[2];
                                wave.freq_dig[2] =
wave.freq_dig[1];
                                wave.freq_dig[1] = num;
                                wave.freq = ((wave.freq*10) +
num);
                                four_flag  = 1;
                        }
                        break;
                default:
                        i = 0; // if i is not 0,1,2,3 or 4,
assign it to zero
                        break;
                }
        print_digit(wave.freq_dig[4]);
        GPIOA->ODR &= ~(1U << 11); //reset  a5
        delay_ms(2);
        GPIOA->ODR |= (1U << 11);  //set  a5

        print_digit(wave.freq_dig[3]);
        GPIOA->ODR &= ~(1U << 12); //reset  a4
```

```
            delay_ms(2);
            GPIOA->ODR |= (1U << 12);  //set  a4

            print_digit(wave.freq_dig[2]);
            GPIOA->ODR &= ~(1U << 5);  //reset  a3
            delay_ms(2);
            GPIOA->ODR |= (1U << 5);   //set  a3

            print_digit(wave.freq_dig[1]);
            GPIOB->ODR &= ~(1U << 9);  //reset  d10
            delay_ms(2);
            GPIOB->ODR |= (1U << 9);   //set  d10
        }
}
void EXTI2_3_IRQHandler(void)//interrupt function for keypads first and last columns
{
    //reset ssd pins to have a clear look
    GPIOA->ODR |= (1U << 11);  //set  a5
    GPIOA->ODR |= (1U << 12);  //set  a4
    GPIOA->ODR |= (1U << 5);   //set  a3
    GPIOB->ODR |= (1U << 9);   //set  d10
    /*handles first and last columns*/
    enter_flag = 0;
    GPIOB->ODR &= ~(1U << 9);  //reset  d10
    if((GPIOB->IDR >> 3) & 1){
        clear_rows_keypad();
```

```
//try for each keypad rows


GPIOB->ODR |= (1U << 6); //keypad A button
if((GPIOB->IDR >> 3) & 1){
      print_flag = 0; //get out printing
      amplitude_flag = 1;
      frequency_flag  = 0;
      i = 0;
      //print_digit(11); //letter A
      delay_ms(500); //little bit delay for debouncing


}
GPIOB->ODR &= ~(1U << 6); //close first row
GPIOB->ODR |= (1U << 7); //keypad B button
if((GPIOB->IDR >> 3) & 1){


      print_flag = 0; //get out printing
      amplitude_flag = 0;
      frequency_flag  = 1;
      //print_digit(8);
      delay_ms(500); //little bit delay for debouncing
}
GPIOB->ODR &= ~(1U << 7); //close second row
GPIOA->ODR |= (1U << 15); //keypad C button
if((GPIOB->IDR >> 3) & 1){
```

```
                amplitude_flag = 0;
                frequency_flag  = 0;
                //print_digit(12);
                wave.state++;

                if(wave.state > noise){
                     wave.state = sin;
                }
                delay_ms(500); //little bit delay for debouncing
           }
           GPIOA->ODR &= ~(1U << 15); //close third row
           GPIOB->ODR |= (1U << 1); //keypad D button
           if((GPIOB->IDR >> 3) & 1){
                frequency_flag  = 0;
                print_flag = 1;
                print_counter++; //counter for switching between
printing modes(amplitude,freq,wave type)
                delay_ms(500); //little bit delay for debouncing
           }
           GPIOB->ODR &= ~(1U << 1); //close fourth row
           EXTI-> RPR1 |= (1 << 3); //clear pending bit
           set_rows_keypad();
      }
      if((GPIOB->IDR >> 2) & 1){
           clear_rows_keypad();
           //try for each keypad rows
           GPIOB->ODR |= (1U << 6);  //keypad 1 button
```

```
        if((GPIOB->IDR >> 2) & 1){
            print_flag = 0; //get out printing
            num = 1;
            i++;
            //print_digit(1);
            delay_ms(500); //little bit delay for debouncing
        }
        GPIOB->ODR &= ~(1U << 6); //close first row
        GPIOB->ODR |= (1U << 7);  //keypad 4 button
        if((GPIOB->IDR >> 2) & 1){
            print_flag = 0; //get out printing
            num = 4;
            i++;
            //print_digit(4);
            delay_ms(500); //little bit delay for debouncing
        }
        GPIOB->ODR &= ~(1U << 7); //close second row
        GPIOA->ODR |= (1U << 15); //keypad 7 button
        if((GPIOB->IDR >> 2) & 1){
            print_flag = 0; //get out printing
            num  = 7;
            i++;
            //print_digit(7);
            delay_ms(500); //little bit delay for debouncing
        }
        GPIOA->ODR &= ~(1U << 15); //close third row
```

```
		GPIOB->ODR |= (1U << 1);  //keypad * button
		if((GPIOB->IDR >> 2) & 1){
			i++;
			print_flag = 0; //get out printing
			dot_flag = 1;
			dot_print_flag = 1;
			dot_index = i;
			//print_digit(19);
			delay_ms(500); //little bit delay for debouncing
		}
		GPIOB->ODR &= ~(1U << 1); //close fourth row
		EXTI-> RPR1 |= (1 << 2); //clear pending bit
		set_rows_keypad();
	}
}
void EXTI0_1_IRQHandler(void)//interrupt function for keypads
second and third columns
{
	//reset ssd pins to have a clear look
	GPIOA->ODR |= (1U << 11);  //set  a5
	GPIOA->ODR |= (1U << 12);  //set  a4
	GPIOA->ODR |= (1U << 5);   //set  a3
	GPIOB->ODR |= (1U << 9);   //set  d10
	print_flag = 0;//get out from print state
	/*handles second and third columns*/
	GPIOB->ODR &= ~(1U << 9);  //reset  d10
	if((GPIOB->IDR >> 0) & 1){
```

```
clear_rows_keypad();
//try for each keypad rows


GPIOB->ODR |= (1U << 6);  //keypad 3 button
if((GPIOB->IDR >> 0) & 1){
     enter_flag = 0;
     //print_digit(3);
     num  = 3;
     i++;
     delay_ms(500); //little bit delay for debouncing
}
GPIOB->ODR &= ~(1U << 6); //close first row
GPIOB->ODR |= (1U << 7);  //keypad 6 button
if((GPIOB->IDR >> 0) & 1){
     enter_flag = 0;
     //print_digit(6);
     num = 6;
     i++;
     delay_ms(500); //little bit delay for debouncing
}
GPIOB->ODR &= ~(1U << 7); //close second row
GPIOA->ODR |= (1U << 15);  //keypad 9 button
if((GPIOB->IDR >> 0) & 1){
     //print_digit(9);
     enter_flag = 0;
     num = 9;
```

```
			i++;
			delay_ms(500); //little bit delay for debouncing
		}
		GPIOA->ODR &= ~(1U << 15); //close third row
		GPIOB->ODR |= (1U << 1);  //keypad # button
		if((GPIOB->IDR >> 0) & 1){
			amplitude_flag = 0;
			frequency_flag = 0;
			enter_flag = 1;
			//print_digit(15);
			delay_ms(500); //little bit delay for debouncing
		}
		GPIOB->ODR &= ~(1U << 1); //close fourth row
		EXTI-> RPR1 |= (1 << 0); //clear pending bit
		set_rows_keypad();
	}
	if((GPIOA->IDR >> 1) & 1){
		clear_rows_keypad();
		//try for each keypad rows

		GPIOB->ODR |= (1U << 6);  //keypad 2 button
		if((GPIOA->IDR >> 1) & 1){
			enter_flag = 0;
			//print_digit(2);
			num = 2;
			i++;
```

```
            delay_ms(500); //little bit delay for debouncing
        }
        GPIOB->ODR &= ~(1U << 6); //close first row
        GPIOB->ODR |= (1U << 7);  //keypad 5 button
        if((GPIOA->IDR >> 1) & 1){
            enter_flag = 0;
            //print_digit(5);
            num = 5;
            i++;
            delay_ms(500); //little bit delay for debouncing
        }
        GPIOB->ODR &= ~(1U << 7); //close second row
        GPIOA->ODR |= (1U << 15);  //keypad 8 button
        if((GPIOA->IDR >> 1) & 1){
            enter_flag = 0;
            //print_digit(8);
            num = 8;
            i++;
            delay_ms(500); //little bit delay for debouncing
        }
        GPIOA->ODR &= ~(1U << 15); //close third row
        GPIOB->ODR |= (1U << 1);  //keypad 0 button
        if((GPIOA->IDR >> 1) & 1){
            //print_digit(0);
            enter_flag = 0;
            num = 0;
```

```
                i++;

                delay_ms(500); //little bit delay for debouncing

            }

            GPIOB->ODR &= ~(1U << 1); //close fourth row

            EXTI-> RPR1 |= (1 << 1); //clear pending bit

            set_rows_keypad();

        }

}

void TIM2_IRQHandler(void) {

    // update duty (CCR2)

        uint32_t bol = (245*(wave.freq+1));

         TIM2->PSC = (SYSTEM_CLK / bol);

        switch(wave.state){

        case sin:

            TIM2 -> CCR1
=(float)(wave.amp/3.3)*(lookup_table[0][tim_counter]);

            break;

        case square:

            TIM2 -> CCR1 =(float) (wave.amp/3.3)*
lookup_table[1][tim_counter];

            break;

        case triangle:

            TIM2 -> CCR1 = (float)
(wave.amp/3.3)*lookup_table[2][tim_counter];

            break;

        case sawtooth:

            TIM2 -> CCR1 =(float)
(wave.amp/3.3)*lookup_table[3][tim_counter];
```

```
            break;
        case noise:
            TIM2 -> CCR1 = rand() % 1024;
            break;
        }
        tim_counter++;
        if(tim_counter>128){
            tim_counter = 0;
        }
    // Clear update status register
            TIM2->SR &= ~(1U << 0);
}
void init_pwm(){
    // Enable TIM2 clock
    RCC->APBENR1 |= RCC_APBENR1_TIM2EN;
    // Set alternate function to 2
    // 0 comes from PA0
    GPIOA->AFR[0] |= (2U << 4*0);
    // Select AF from Moder
    setMode('A',0,'F');
    //I NEED TIM2 CH 1
    // zero out the control register just in case
    TIM2->CR1 = 0;
    // Select PWM Mode 1
    TIM2->CCMR1 |= (6U << 4);
    // Preload Enable
```

```
    TIM2->CCMR1 |= TIM_CCMR1_OC1PE;

    // Capture compare ch1 enable

    TIM2->CCER |= TIM_CCER_CC1E;

    // zero out counter

    TIM2->CNT = 0;

    //initialize the frequency

    TIM2->PSC = 100;

    TIM2->ARR = 245;

    // zero out duty

    TIM2->CCR1 = 0;

    // Update interrupt enable

    TIM2->DIER |= (1 << 0);

    // TIM2 Enable

    TIM2->CR1 |= TIM_CR1_CEN;

    NVIC_SetPriority(TIM2_IRQn, 3);

    NVIC_EnableIRQ(TIM2_IRQn);

}

int main(void) {

     init_systick(SystemCoreClock/1000);

/*open clocks*/

     openClock('A');

     openClock('B');

     /*configure 7 segment pins*/

     setMode('A',8,'O');

     setMode('A',4,'O');

     setMode('A',5,'O');
```

```
	setMode('A',12,'O');

	setMode('A',6,'O');

	setMode('A',7,'O');

	setMode('A',11,'O');

	setMode('B',4,'O');

	setMode('B',5,'O');

	setMode('B',8,'O');

	setMode('B',9,'O');


	//set ssd digits high as initial to close all digits

	GPIOA->ODR |= (1U << 11);  //set  a5

	GPIOA->ODR |= (1U << 12);  //set  a4

	GPIOA->ODR |= (1U << 5);   //set  a3

	GPIOB->ODR |= (1U << 9);   //set  d10


	/*configure keypad*/

	//rows are output, columns are input

	config_keypad_pins();//configure the pins

	config_keypad_IRQs();//configure the interrupts


	//initialize the wave

	wave.state = square;

	init_pwm();

while(1){

	if(print_flag == 1){//print the wave mode if the D button is
pushed

		switch(print_counter){
```

```
            case 0:
                  print_amplitude();
                  break;
            case 1:
                  print_mode();
                  break;
            case 2:
                  print_frequency();
                  break;
            default:
                  print_counter = 0;
                  break;
            }
      }
      if(amplitude_flag == 1){ //switch to amplitude mode
            num = 0;
            set_amplitude();
      }
      if(frequency_flag == 1){
            num = 0;
            set_frequency();
      }
         //TIM2->ARR = (uint32_t)(16000000/(8*(wave.freq+1)));//8
comes from arr
}
return 0;
}
```

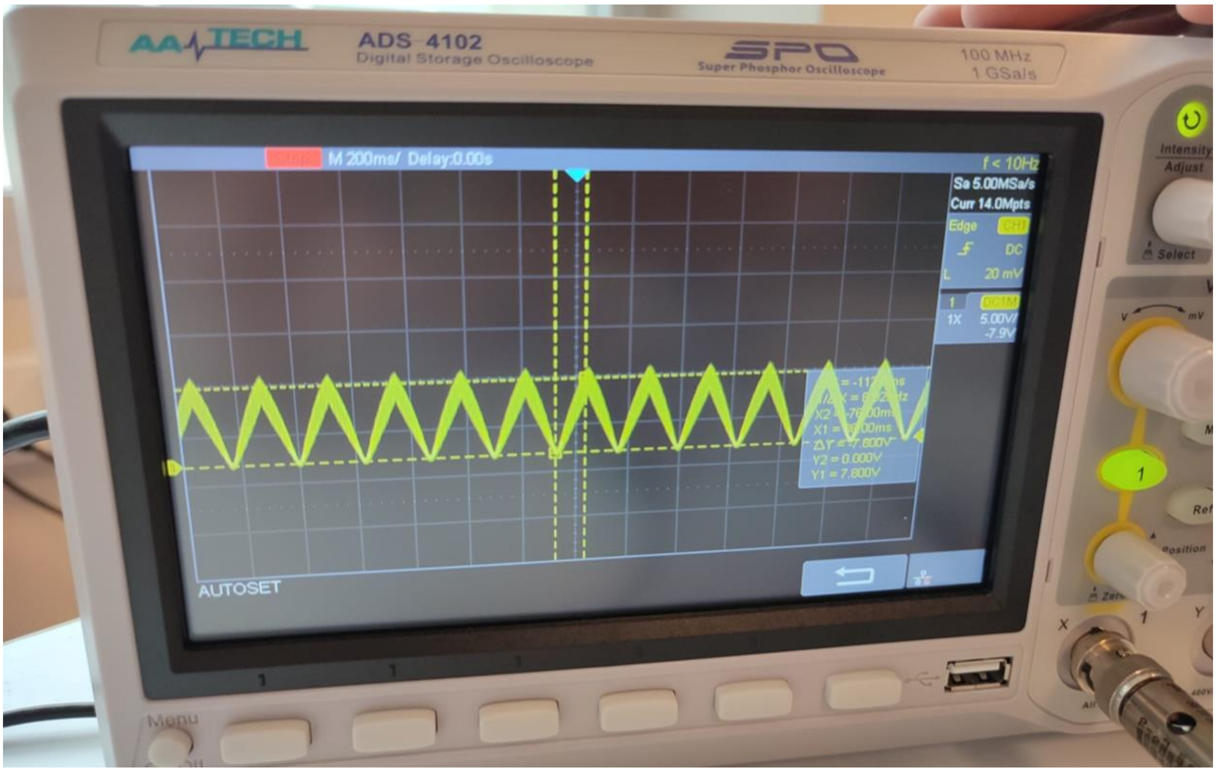

Figure 4: Triangle Wave

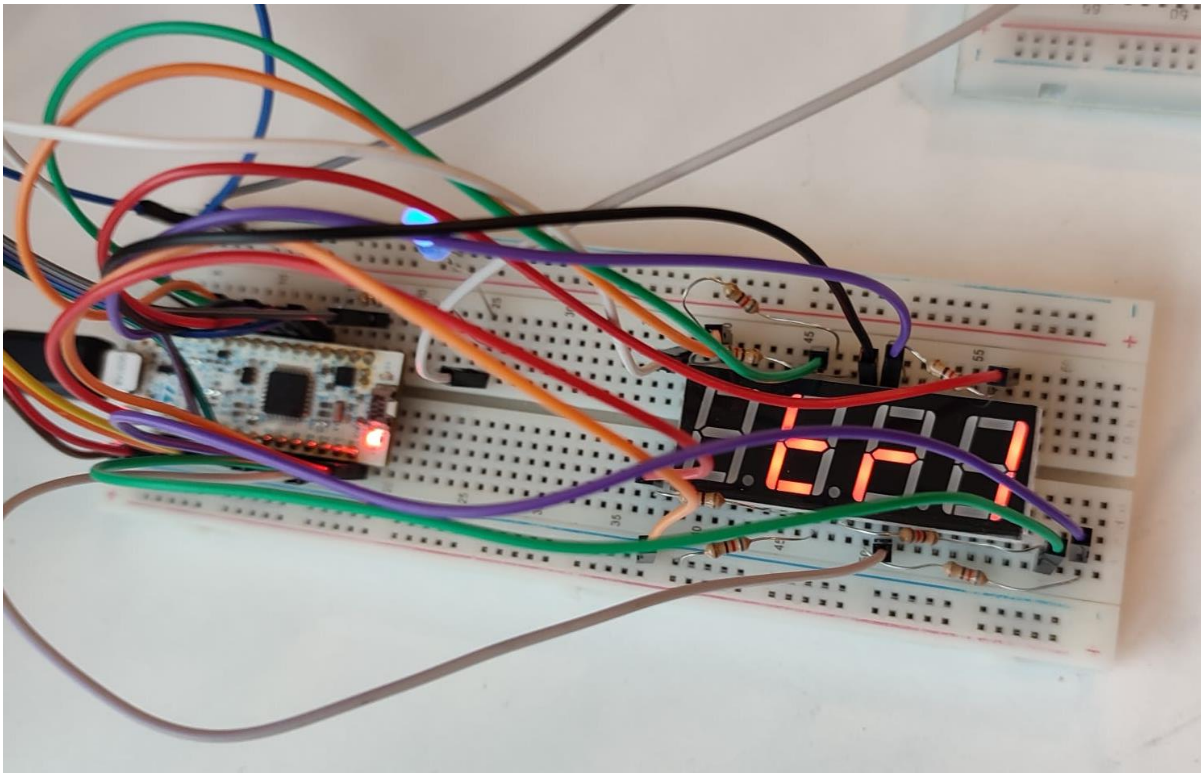

Figure 5: Triangle Mode

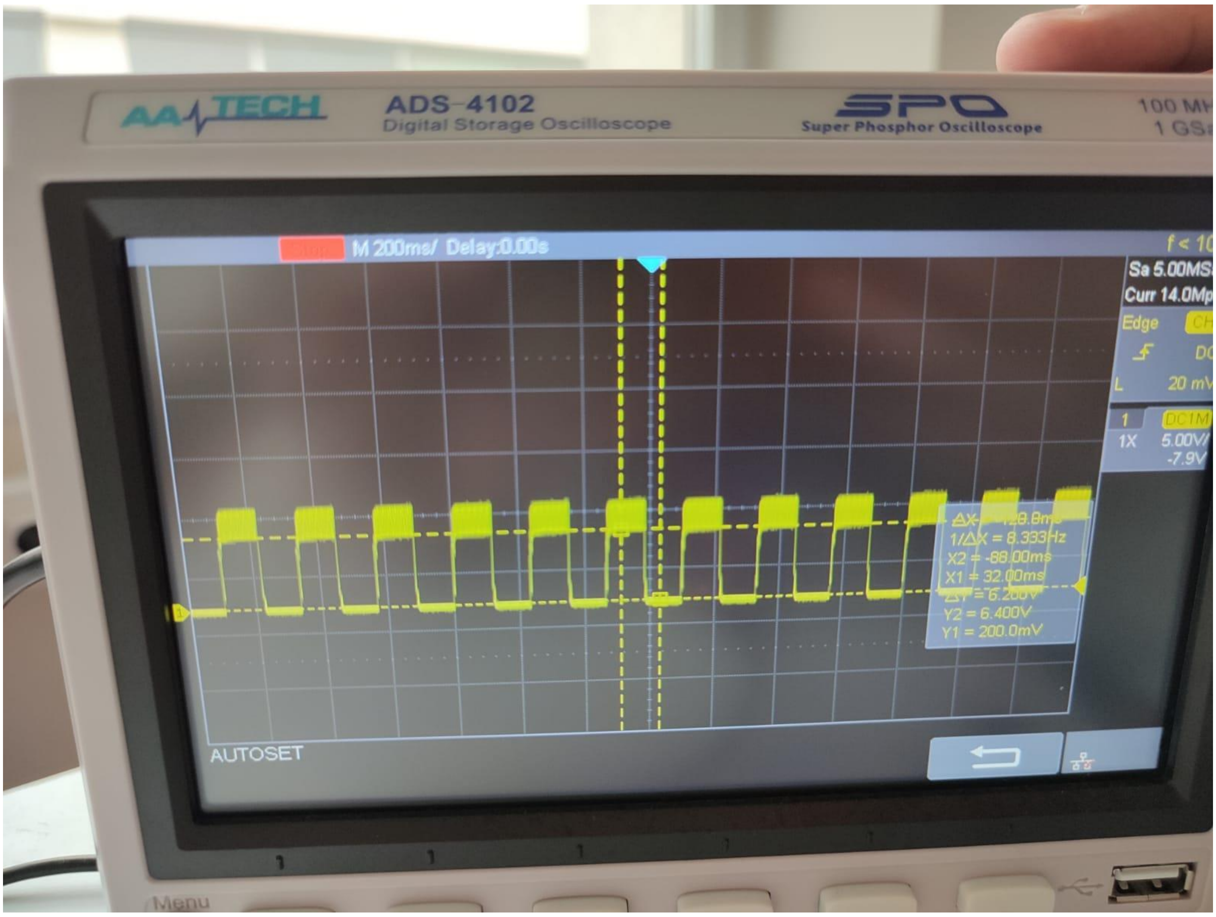

Figure 6: Square Wave

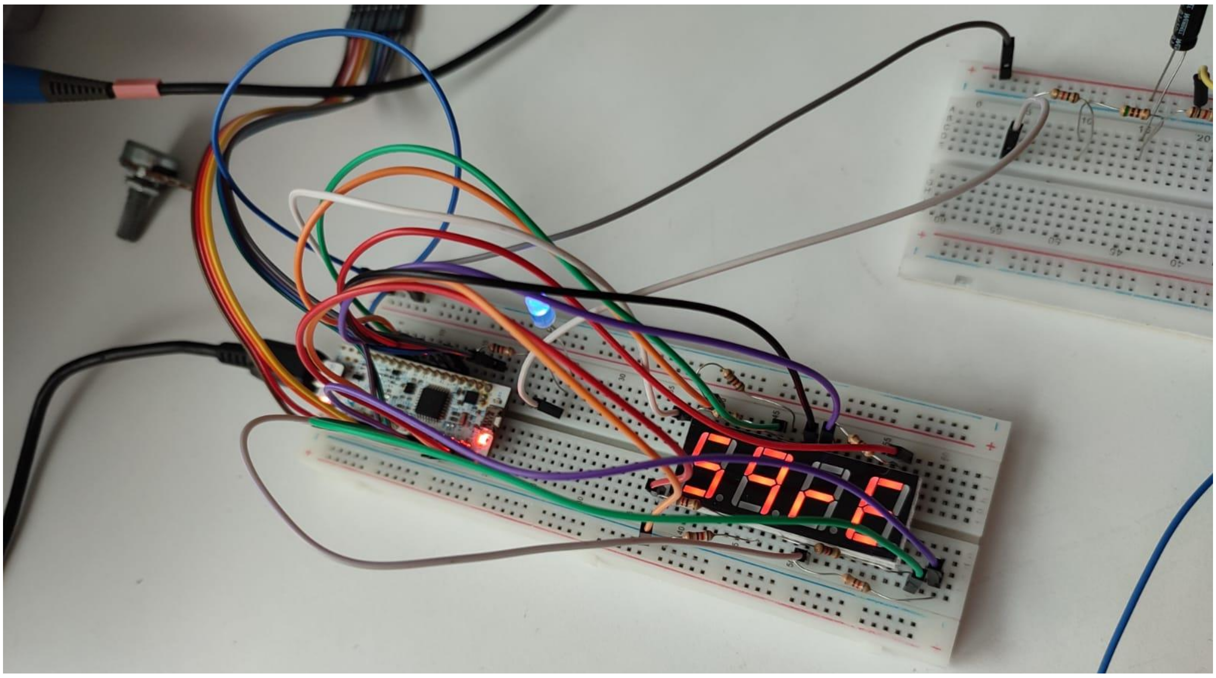

Figure 7: Square Mode

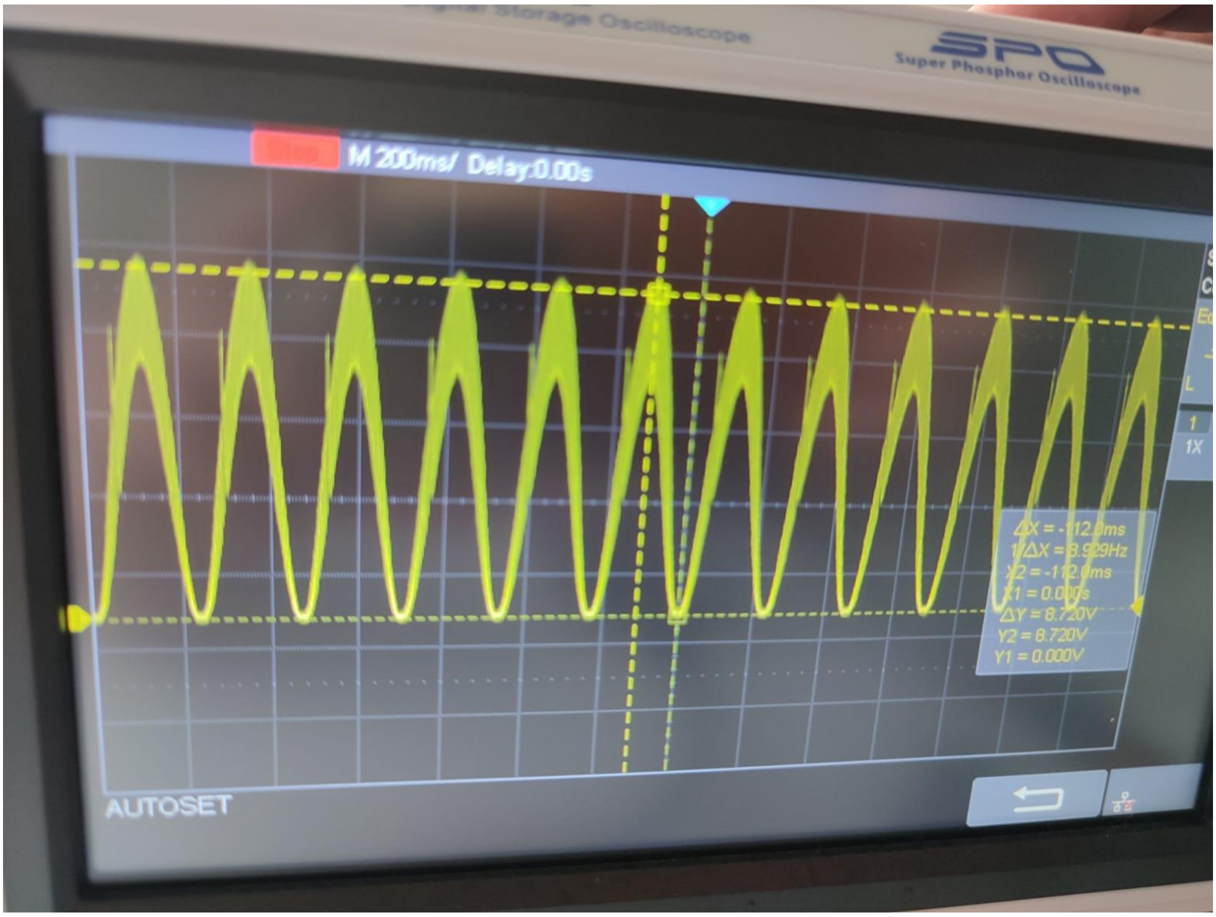

Figure 8: Sinus Wave

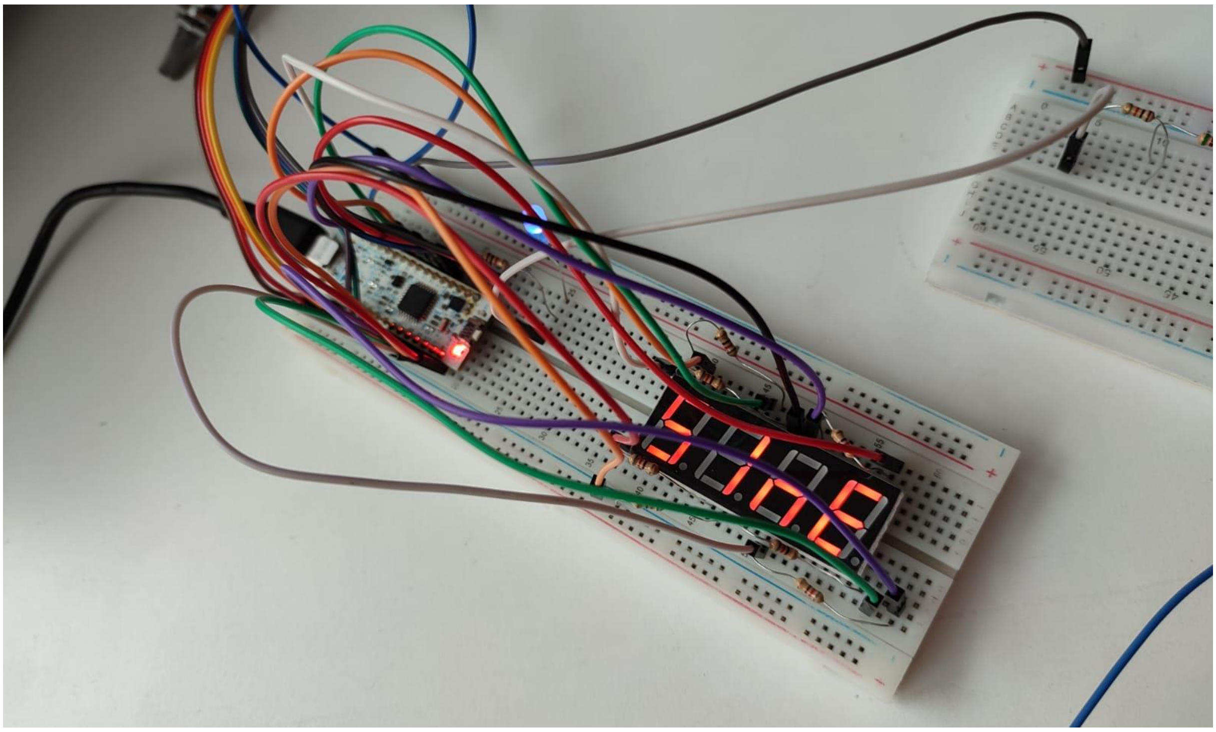

Figure 9: Sinus Mode

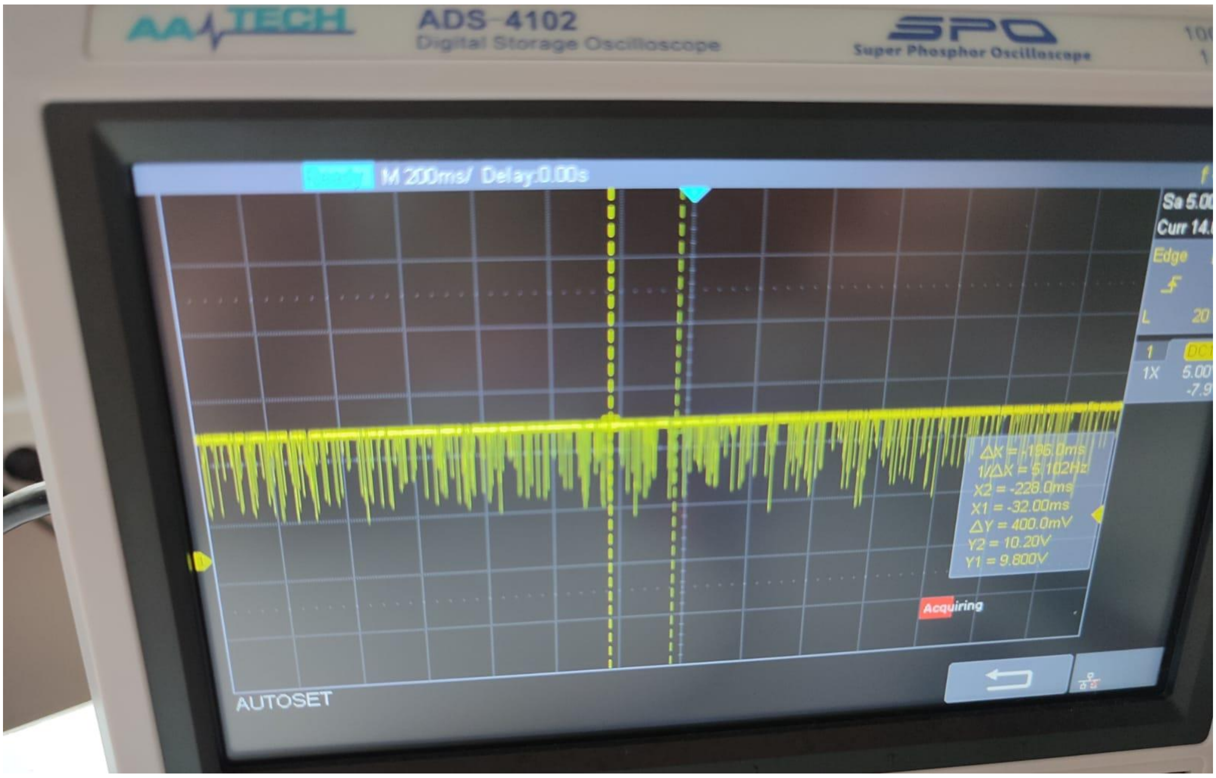

Figure 10: Noise Wave

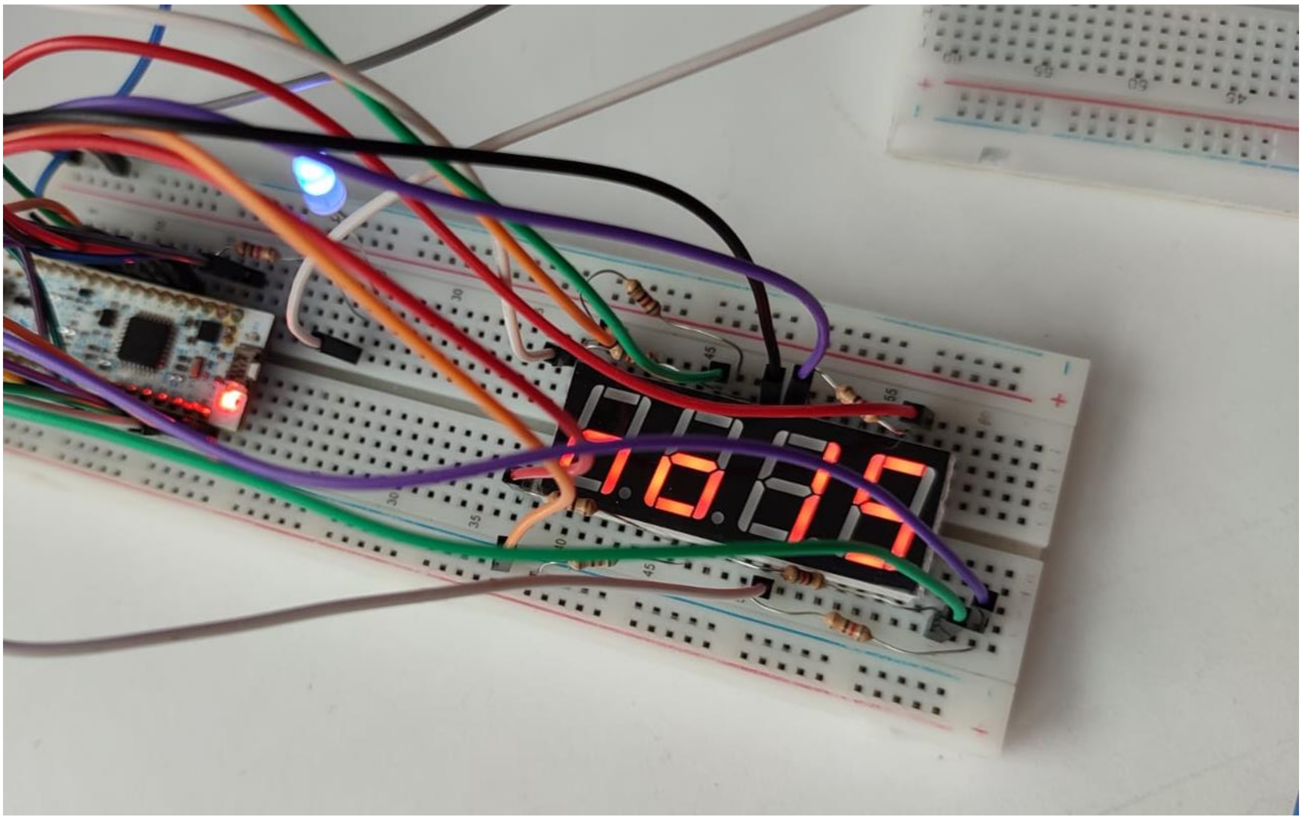

Figure 11: Noise Mode

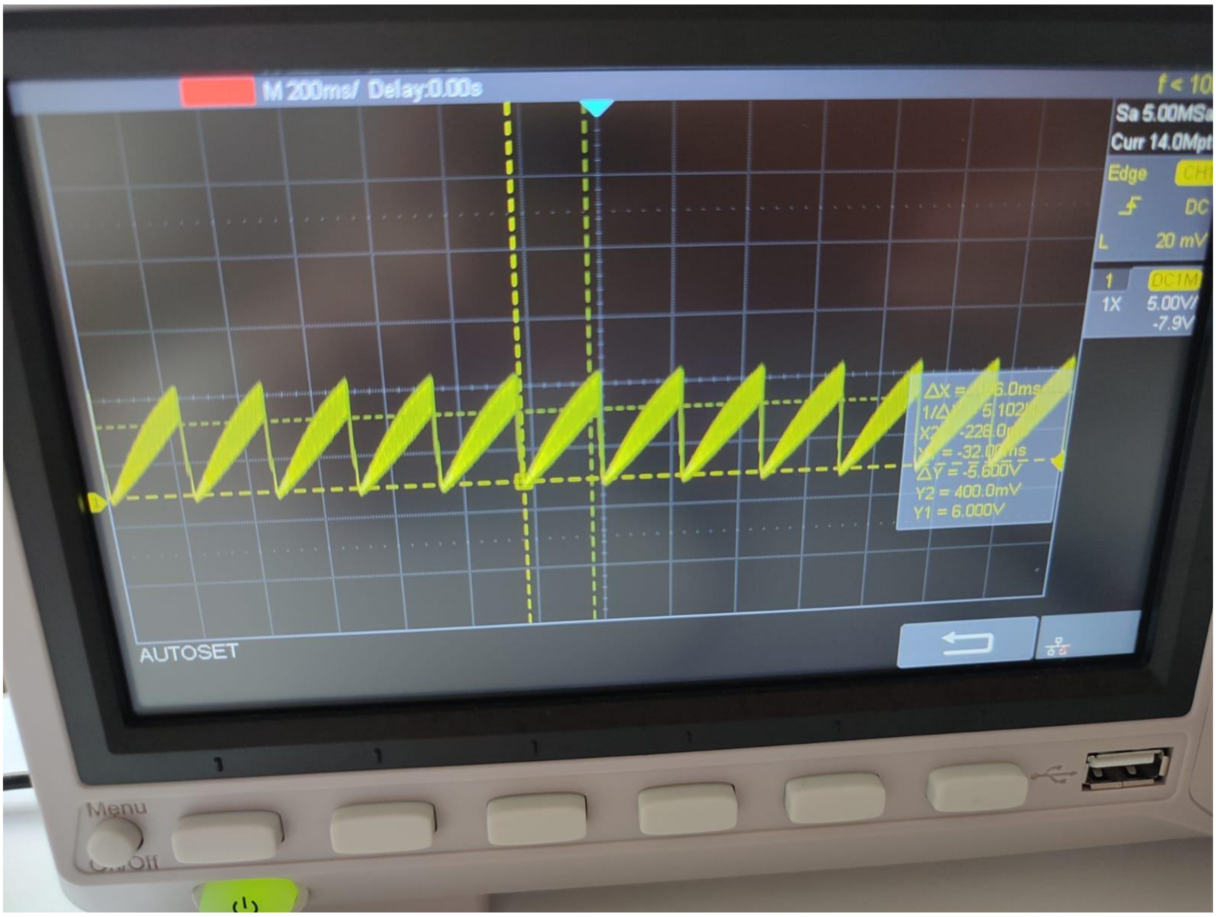

Figure 12: Sawtooth Wave

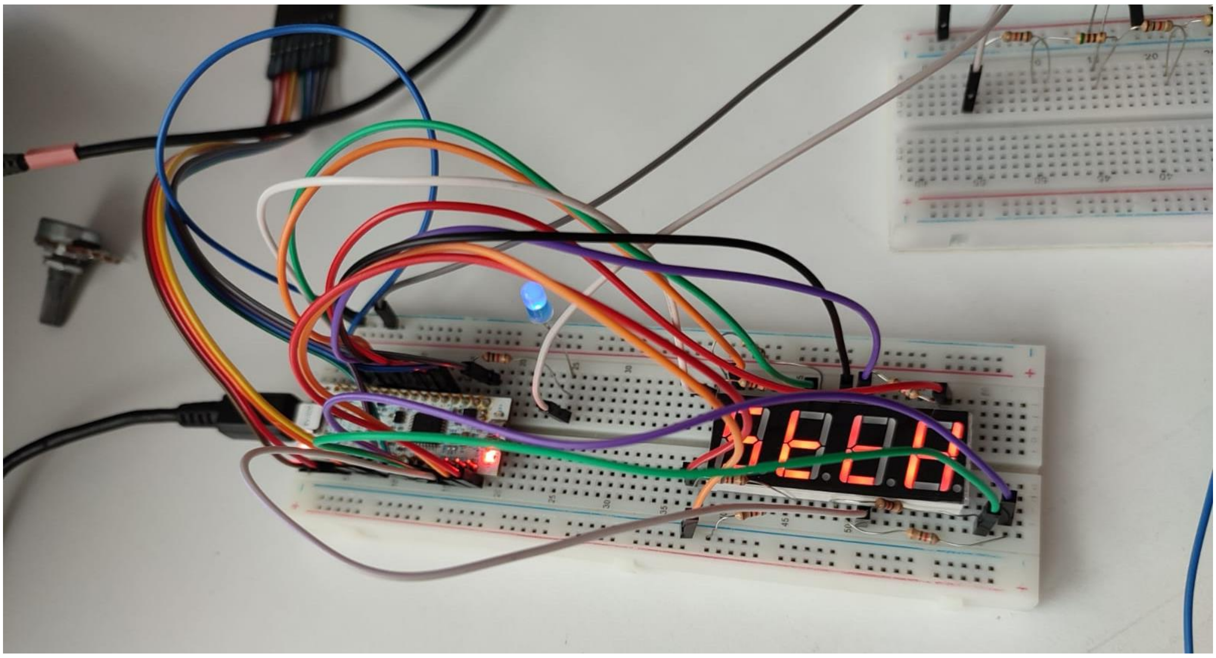

Figure 13: Sawtooth Mode

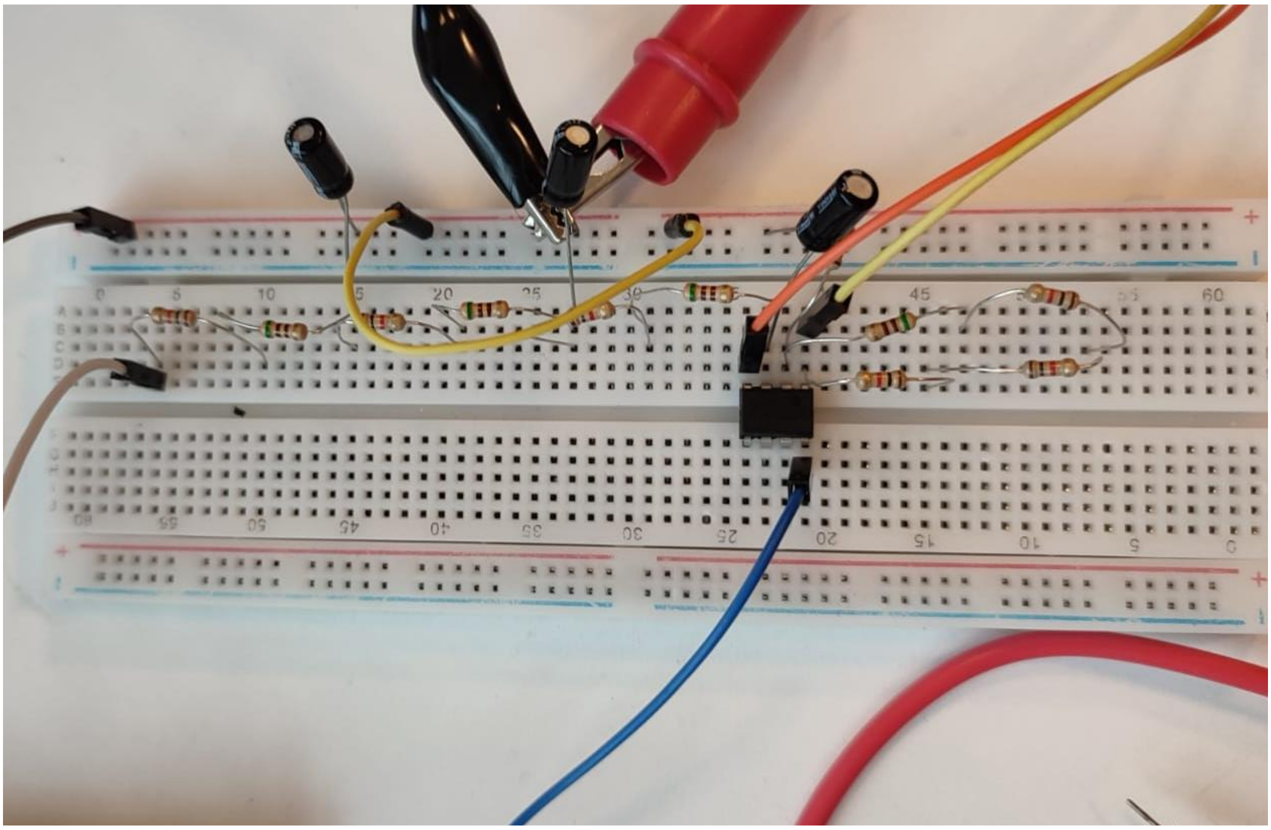

Figure 14: Filter Circuit

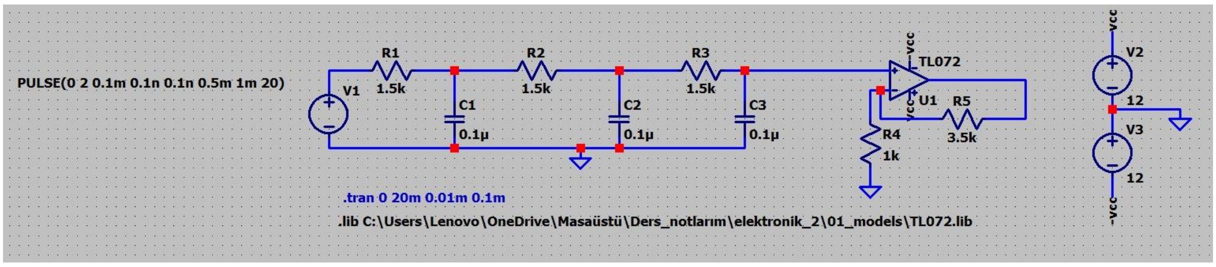

Figure 15: Filter Simulation

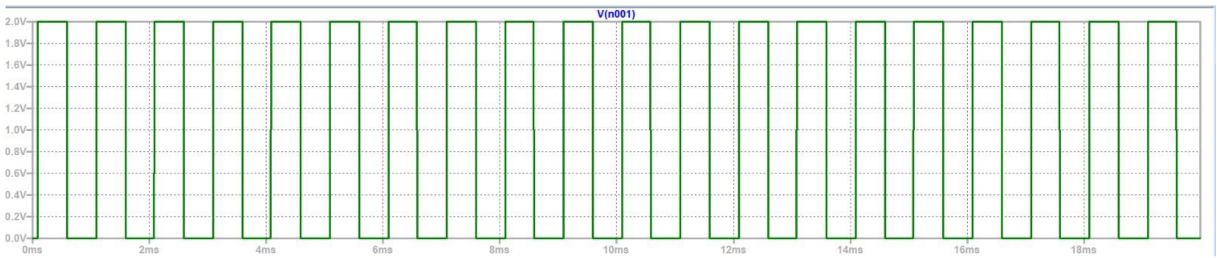

Figure 16: Input Signal

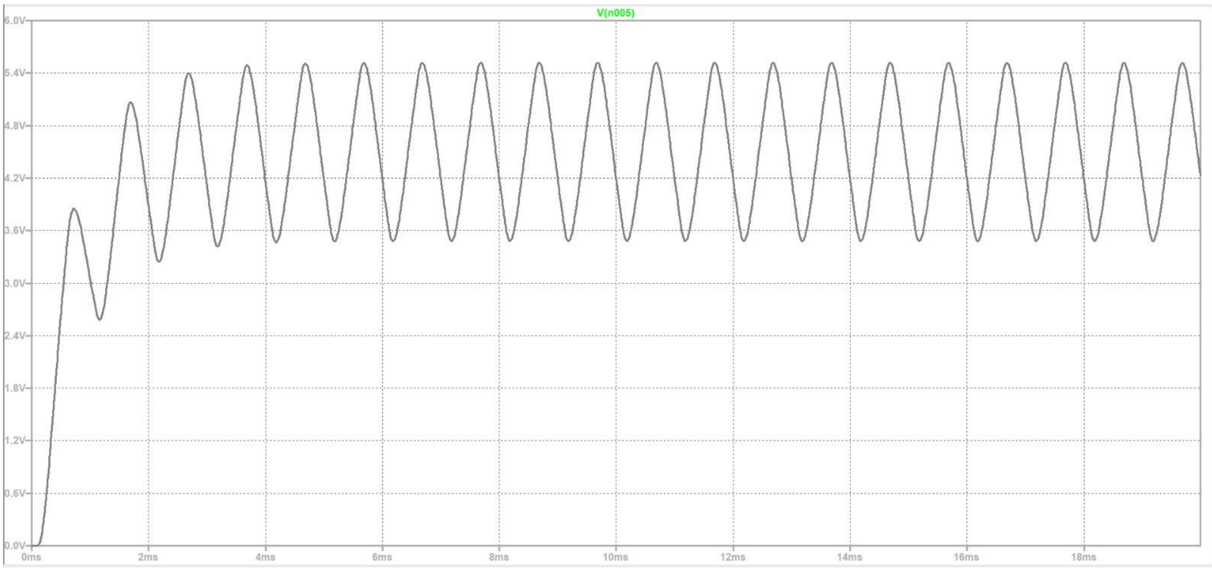

Figure 17: Output Signal

Conclusion

The code written for the project rotates in an endless while loop within the main function. It enters the relevant interrupt according to pressing the A, B, C or D buttons and sets the related amplitude, frequency or mode change flags to their new values. According to these flag values, it enters the relevant if blocks inside the main function and run the desired functions. These functions assign the values entered on the membrane keypad to the relevant variables, and by using these variables, the characteristics of the signal printed on the pwm can be changed. Sample tables were created for each desired signal. The values in these tables are given to the pwm output one by one, and by changing the duty cycle values of the pwm signal according to these values, the basis for obtaining the desired type of signal output is prepared. The desired signals are displayed on the oscilloscope screen by connecting a filter to the output of the pin whose Pwm output is taken.

Theoretically, the rc filter outputs 10% of the input signal. In this process, the cutoff frequency is determined as 1khz. The selected values are the resistance 1500 ohm and the capacitance value 0.1 microfarad. Rc filter has 3 layers. TL 072 opamp is used outside of this filter. The resistors used for gain were first increased 10 times from the 1+R2/R1 calculation. However, in the simulation, the result did not produce the same input and output. Then, by trial and error method, r2 was determined as 3.5k, r1 1k. In the simulation, these values and the input and output values are the same. However, it was observed that the output of the opamp was higher than the given amplitude when the measurements were taken.

Video Link:

https://youtu.be/A0BqAIYUYwI