GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC335 – MICROPROCESSORS LABAORATORY

LAB 6

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

Problem 1

In this problem, you will be working on implementing **a signal follower**. Attach a signal to one of the pins, capture its value and replay it back using PWM. You should see the original signal back on the oscilloscope.

- You can use a function generator to send the signal.
- The signal frequency depends on how fast you can go.

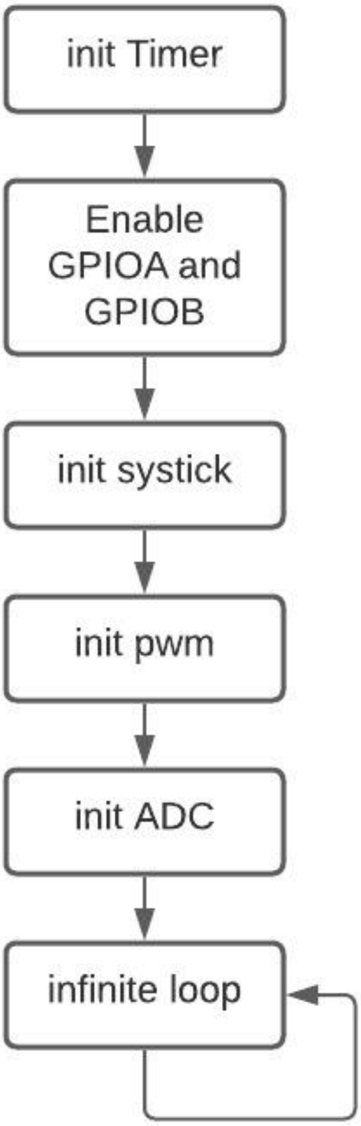


Figure 1: Problem 1 Flowchart

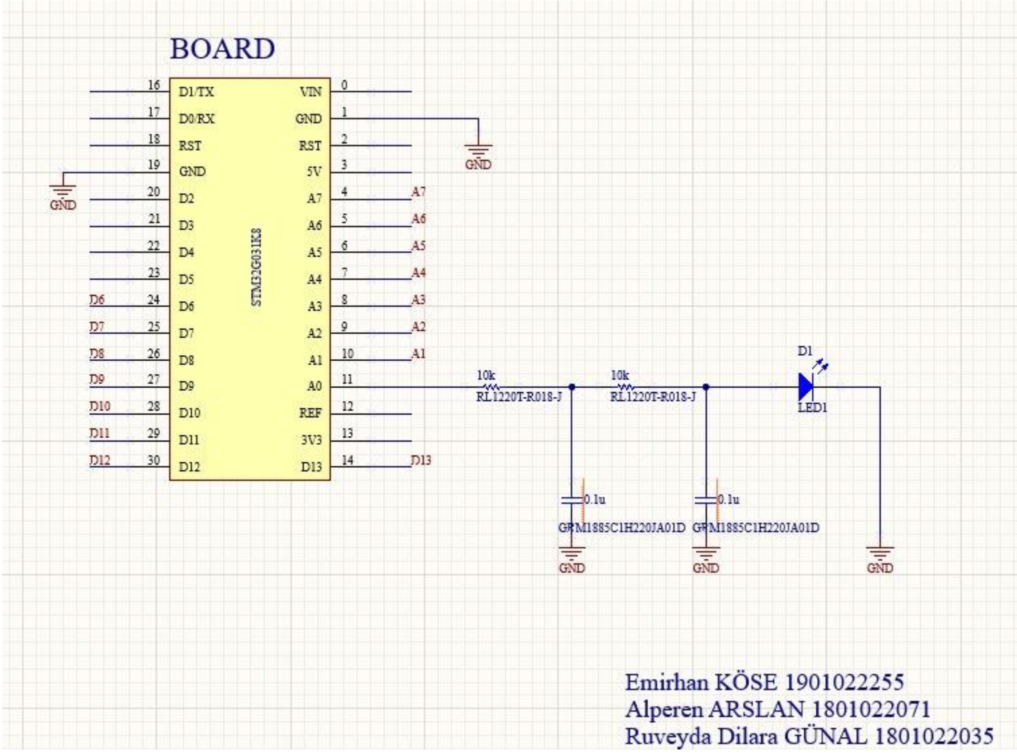


Figure 2: Problem 1 Block Diagram

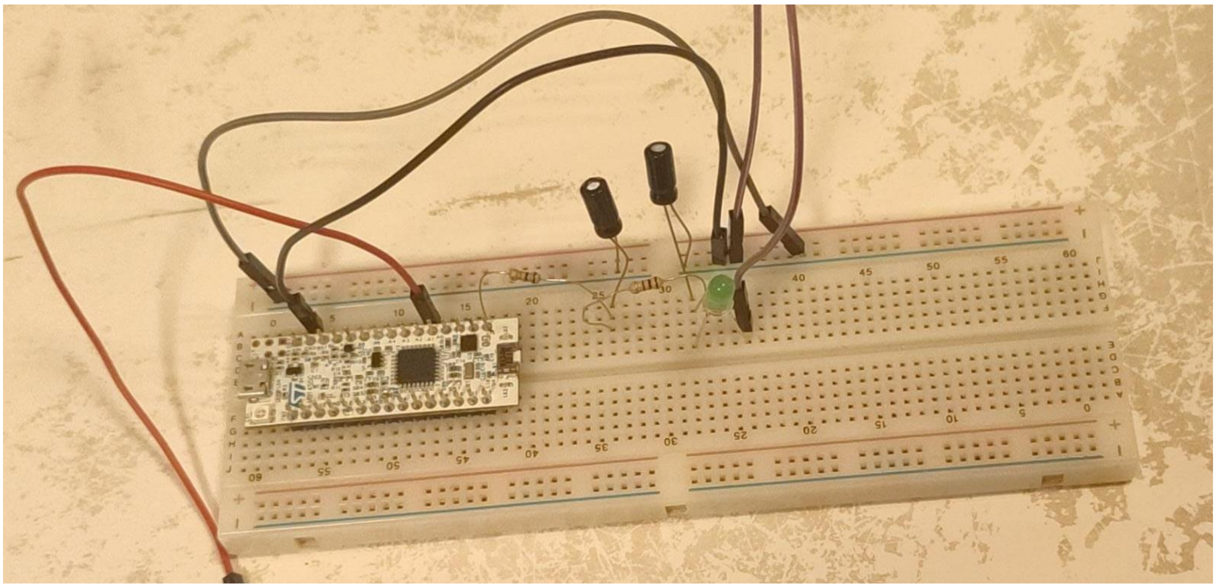


Figure 3: Problem 1 Circuit

main

```
#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;
}
```

c

```
#include "BSP.h"
#include "stm32g0xx.h"
#include "system_stm32g0xx.h"

static uint32_t tDelay;
extern uint32_t SystemCoreClock;

/*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_84(){            //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 = 84000000;


}


/*Interrupts*/

void EXTI0_1_IRQHandler(void){


               GPIOA-> ODR |= (1U << 6);//open the led on the PA0


        EXTI-> RPR1 |= (1 << 0); //clear pending bit

}


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);
```

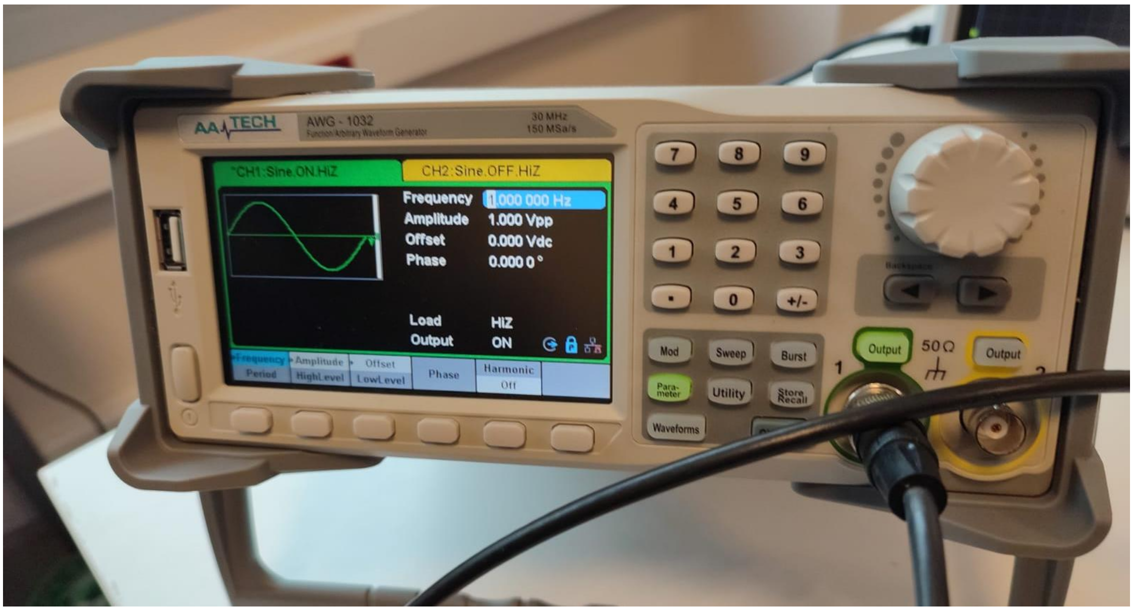
```
}


/*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);
}
```

h

```
#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
#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 EXTI_A0_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);



#endif /* BSP_H_ */
```



Frequency 1.000 000 Hz
Amplitude 1.000 Vpp
Offset 0.000 Vdc
Phase 0.000 0 °
Load HiZ
Output ON

Figure 4: Signal Sent

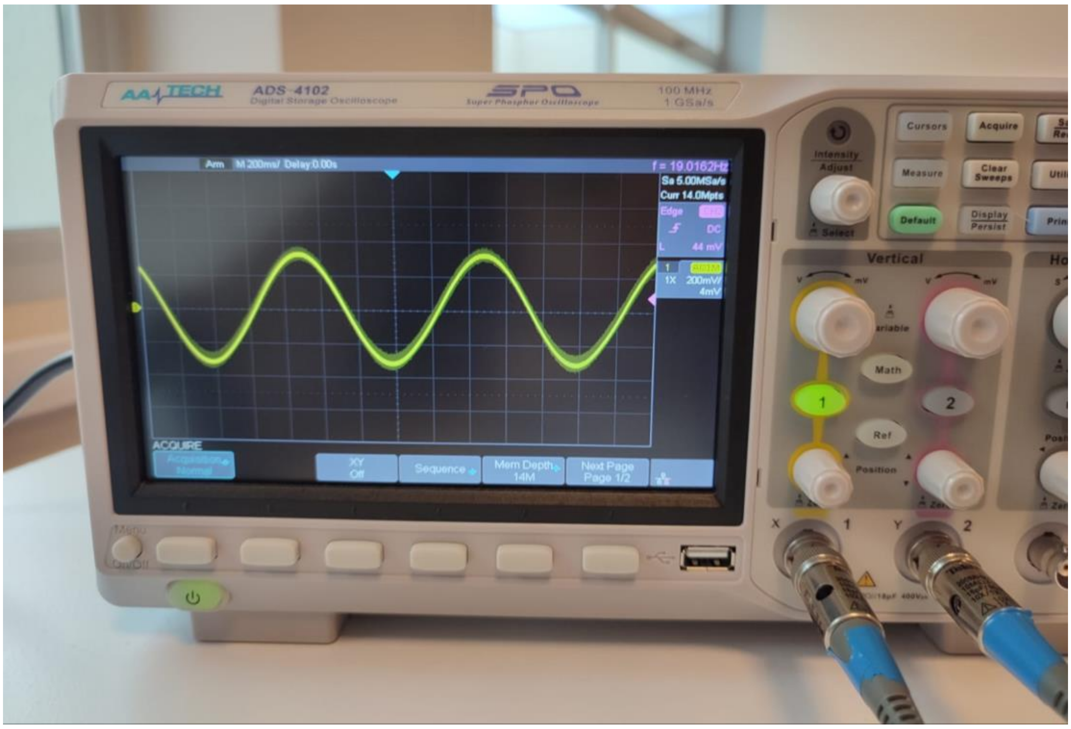


Figure 5: Signal Sent

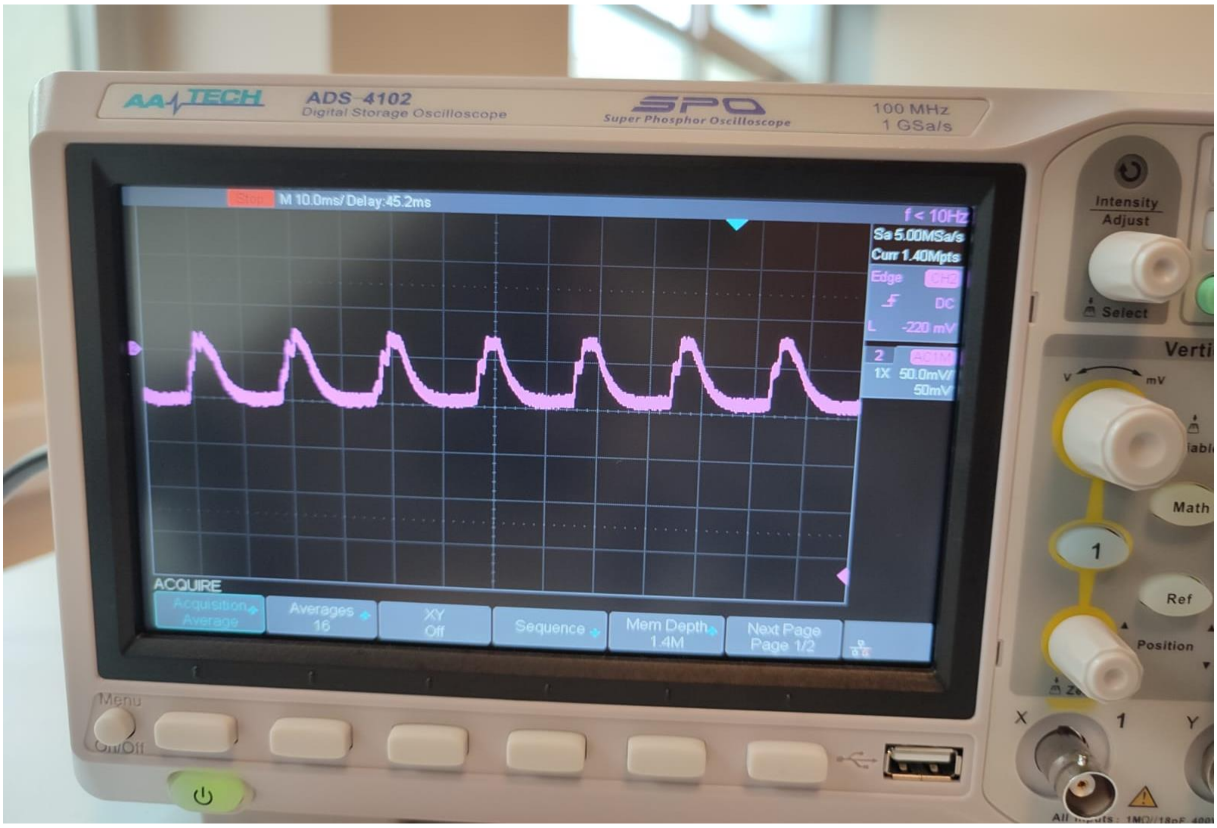


Figure 6: Output Signal

Problem 2

In this problem, you will be working with reading and logging MPU6050 IMU sensor data utilizing Timer, I2C, and UART modules and using MPU6050, and 24LC512 EEPROM.

- Write your I2C routines to read / write multiple data. You should have four functions: single read, single write, multi read, multi write. Multi read and write deal with multiple bytes.
- Write a data structure to hold the sensor data.
- To ensure the data is correct, read all sensor data and send them all over UART to PC as the example below:

  AX: 0.12, AY: 0.53, ..., GX: 1.32, ...

- Sample the sensors every 10 ms, and write the data values to EEPROM. You should first start with writing and reading single bytes. Once the operation is completed successfully, work on your way to write and read multiple bytes.
- EEPROM and MPU6050 should be sharing the same I2C bus. Check the IMU board for pull-up resistors. If it includes pull-up resistors, you should not need to add another set of pull-up resistors.
- Once you press an external button, data collection should start, and once it collects 10 seconds of data, it should stop, and an LED should light up to display data that is ready on EEPROM.
- When the LED is on (meaning there is data on the EEPROM) pressing the button will transmit all the data over UART to your PC.
- You save this to a file and plot your results using Python or MATLAB.

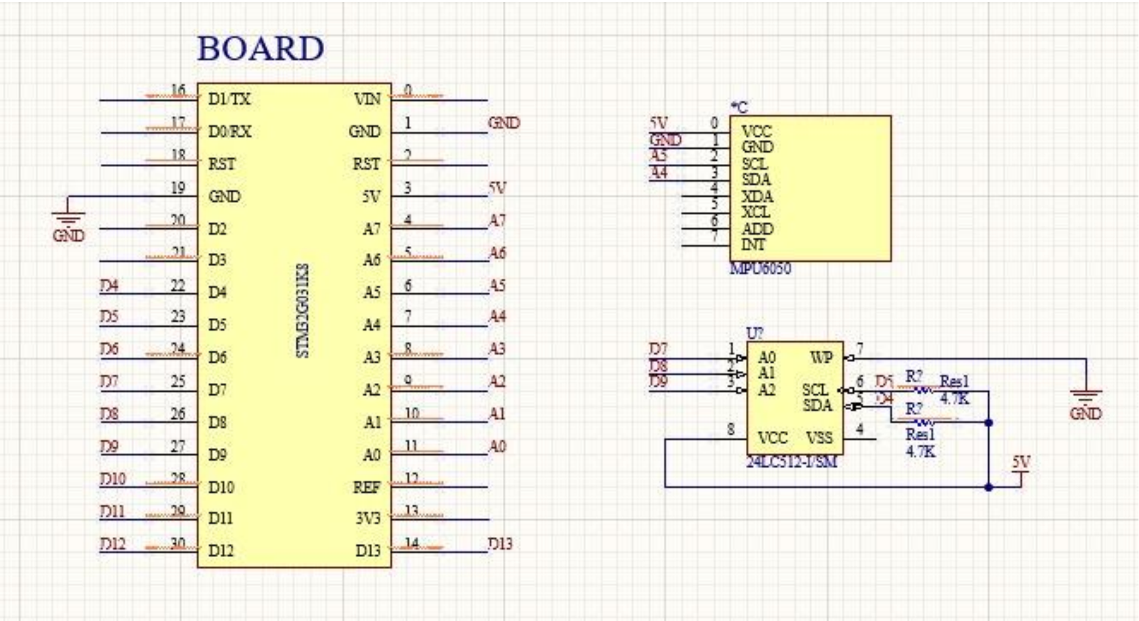


Figure 7: Problem 2 Flowchart

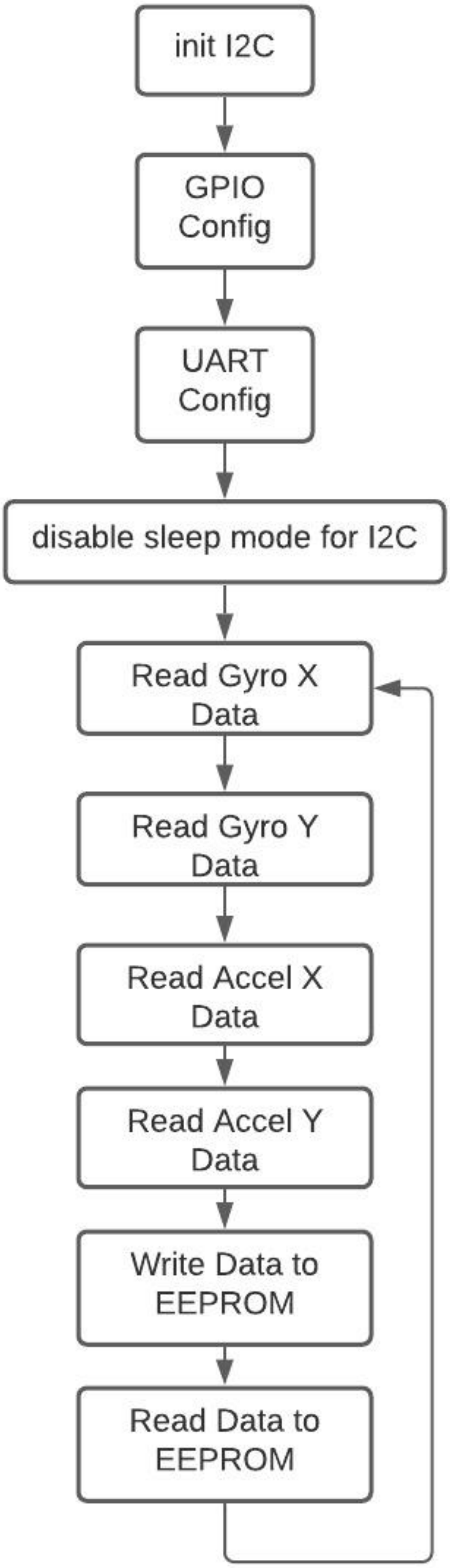


Figure 8: Problem 2 Flowchart

```
#include "stm32g0xx.h"
#include <stdio.h>
// I2C busses == PB8-PB9
#define MPU6050_ADDRESS 0x68
#define MPU6050_PWR_MGMT_1 0x6B
#define MPU6050_ACCEL_XOUT_H 0x3B
#define MPU6050_ACCEL_XOUT_L 0x3C
#define MPU6050_ACCEL_YOUT_H 0x3D
#define MPU6050_ACCEL_YOUT_L 0x3E
#define MPU6050_GYRO_XOUT_H 0x43
#define MPU6050_GYRO_XOUT_L 0x44
#define MPU6050_GYRO_YOUT_H 0x45
#define MPU6050_GYRO_YOUT_L 0x46


struct MPU6050_DATA{
  float accel_x;
  float accel_y;
  float gyro_x;
  float gyro_y;
}MPU6050;

void GPIO_Config(void);
void print(char *buf);
void printChar(uint8_t c);
```

```
int _write(int fd, char *ptr, int len);
void USART_Config(uint16_t baud);
void delay(uint32_t s);
void _read_I2C(uint8_t devAddr,uint16_t memAddr, uint8_t *data, int
size);
void _write_I2C(uint8_t devAddr,uint16_t memAddr, uint8_t *data,
int size);
uint8_t read_I2C(uint8_t devAddr, uint8_t regAddr);
void write_I2C(uint8_t devAddr, uint8_t regAddr, uint8_t data);
void init_I2C(void);
void multi_Read_I2C(uint8_t devAddr, uint8_t regAddr, uint8_t
*data, uint32_t num);


void I2C1_IRQHandler(void){


  //only enters when error
}


int main(void) {
  uint16_t data;
  uint16_t EEPROM_MEMORY = 0x00;
  uint8_t EEPROM_ADDRESS = 0x50;


  uint16_t gyro_x;
  uint16_t accel_x;
  uint16_t gyro_y;
  uint16_t accel_y;
  uint8_t MPU6050_data[4];
  uint8_t data_is_back[4];

  init_I2C();
  GPIO_Config();
  USART_Config(9600);



  write_I2C(MPU6050_ADDRESS, MPU6050_PWR_MGMT_1, 0x00); //disable sleep mode for MPU6050
  while(1) {

    data = read_I2C(MPU6050_ADDRESS, MPU6050_GYRO_XOUT_L);
    data = data | (read_I2C(MPU6050_ADDRESS, MPU6050_GYRO_XOUT_H) << 8);
    MPU6050_data[0] = data; //gyro_x
    MPU6050.gyro_x = (float)(data) / (131.0);

    data = read_I2C(MPU6050_ADDRESS, MPU6050_GYRO_YOUT_L);
    data = data | (read_I2C(MPU6050_ADDRESS, MPU6050_GYRO_YOUT_H) << 8);
    MPU6050_data[1] = data; //gyro_y
    MPU6050.gyro_y = (float)(data) / (131.0);

    data = read_I2C(MPU6050_ADDRESS, MPU6050_ACCEL_XOUT_L);
    data = data | (read_I2C(MPU6050_ADDRESS, MPU6050_ACCEL_XOUT_H) << 8);
    MPU6050_data[2] = data; //accel_x
    MPU6050.accel_x = (float)(data) / (16384.0);


    data = read_I2C(MPU6050_ADDRESS, MPU6050_ACCEL_YOUT_L);
    data = data | (read_I2C(MPU6050_ADDRESS, MPU6050_ACCEL_YOUT_H)
<< 8);
    MPU6050_data[3] = data;
    MPU6050.accel_y = (float)(data) / (16384.0);


    /*printf("MPU6050 GYRO_X = %f\r\n",MPU6050.gyro_x);
    delay(10000);
    printf("MPU6050 GYRO_Y = %f\r\n",MPU6050.gyro_y);
    delay(10000);
    printf("MPU6050 ACCEL_X = %f\r\n",MPU6050.accel_x);
    delay(10000);
    printf("MPU6050 ACCEL_Y = %f\r\n",MPU6050.accel_y);
    delay(10000);
    */
    EEPROM_write_I2C(EEPROM_ADDRESS,EEPROM_MEMORY,&MPU6050_data,4);
    delay(100);
    //printf("EEPROM_WRITTEN_DATA: %d, %d , %d
,%d\r\n",MPU6050_data[0],MPU6050_data[1],MPU6050_data[2],MPU6050_da
ta[3]);
    EEPROM_read_I2C(EEPROM_ADDRESS,EEPROM_MEMORY,&data_is_back,4);
    printf("%d, %d , %d
,%d\r\n",data_is_back[0],data_is_back[1],data_is_back[2],data_is_ba
ck[3]);
    EEPROM_MEMORY += 4;
    delay(1000000);
    }
    return 0;
}

void init_I2C(void){
  RCC->IOPENR |= (1U << 1); //Enable GPIOB
  //Setup PB8 as AF6
  GPIOB->MODER &= ~(3U << 2*8);
  GPIOB->MODER |= (2 << 2*8);
  GPIOB->OTYPER |= (1U << 8);
  //Choose AF from mux
  GPIOB->AFR[1] &= ~(0xFU << 4*0); //High register
  GPIOB->AFR[1] |= (6 << 4*0);
  //Setup PB9 as AF6
  GPIOB->MODER &= ~(3U << 2*9);
  GPIOB->MODER |= (2 << 2*9);
  GPIOB->OTYPER |= (1U << 9);
  //Choose AF6 from mux
  GPIOB->AFR[1] &= ~(0xFU << 4*1);
  GPIOB->AFR[1] |= (6 << 4*1);

  RCC->APBENR1 |= (1U << 21); //Enable I2C1
  I2C1->CR1 = 0; //RESET CR1
  I2C1->CR1 |= (1U << 7); //ERR1
  I2C1->TIMINGR |= (3U << 28); //PRESC
  I2C1->TIMINGR |= (0x13U << 0); //SCLL
  I2C1->TIMINGR |= (0xFU << 8); //SCLH
  I2C1->TIMINGR |= (0x2U << 16); //SDADEL
  I2C1->TIMINGR |= (0x4U << 20); //SCLDEL
  I2C1-> CR1 |= (1U << 0); //PE
  NVIC_SetPriority(I2C1_IRQn, 1);
  NVIC_EnableIRQ(I2C1_IRQn);
}

uint8_t read_I2C(uint8_t devAddr, uint8_t regAddr){
  //Write operation (Send address and register to read)
  I2C1->CR2 = 0; //reset control reg2
  I2C1->CR2 |= ((uint32_t)devAddr << 1);//slave address
  I2C1->CR2 |= (1U << 16); //Number of bytes
  I2C1->CR2 |= (1U << 13); //Generate Start
  while(!(I2C1->ISR & (1U << 1))); //TXIS
  I2C1->TXDR = (uint32_t)regAddr;

  while(!(I2C1->ISR & (1U << 6))); //Transmission complete

  //Read operation (read data)
  I2C1->CR2 = 0;
  I2C1->CR2 |= ((uint32_t)devAddr << 1);
  I2C1->CR2 |= (1U << 10); //Read mode
  I2C1->CR2 |= (1U << 16); //Number of bytes
  I2C1->CR2 |= (1U << 15); //NACK=Not acknowledge
  I2C1->CR2 |= (1U << 25); //Autoend
  I2C1->CR2 |= (1U << 13); //Generate Start
  while(!(I2C1->ISR & (1U << 2)));//wait until RXNE=1

  uint8_t data = (uint8_t)I2C1->RXDR;
  return data;
}

void write_I2C(uint8_t devAddr, uint8_t regAddr, uint8_t data){
  //Write operation (Send address and register to read)
    I2C1->CR2 = 0;
    I2C1->CR2 |= ((uint32_t)devAddr << 1);//slave address
    I2C1->CR2 |= (2U << 16); //Number of bytes
    I2C1->CR2 |= (1U << 25); //AUTOEND
    I2C1->CR2 |= (1U << 13); //Generate Start
    while(!(I2C1->ISR & (1U << 1))); //TXIS
    I2C1->TXDR = (uint32_t)regAddr;
    while(!(I2C1->ISR & (1U << 1))); //TXIS
    I2C1->TXDR = (uint32_t)data;
}
//for MPU6050
// data[0] = regAdress
// data[1] = value for regAddress
//i.e data[0] = MPU6050_PWR_MGMT_1 , data[1] = 0
//i.e write_generel_I2C(MPU6050_ADDRESS, data ,2);
//for 24LC512 EEPROM
// data[0] = regAdress high
// data[1] = regAdress low

// data[2] = value for regAddress

// data[3] = value for regAddress +1

// ...

//i.e write to adress 0x100

//i.e data[0] = 0x1 , data[1] = 0x00 , data[2] = 0

//i.e write_generel_I2C(EEPROM_ADDRESS, data ,3);


/*void write_general_I2C(uint8_t devAddr, uint8_t* data, uint32_t num){

  //Write operation (Send address and register to read)

    I2C1->CR2 = 0;

    I2C1->CR2 |= ((uint32_t)devAddr << 1);//slave address

    I2C1->CR2 |= (num << 16); //Number of bytes

    I2C1->CR2 |= (1U << 25); //AUTOEND

    I2C1->CR2 |= (1U << 13); //Generate Start


    for(int i=0; i<num; i++ ){

      while(!(I2C1->ISR & (1U << 1))); //TXIS

      I2C1->TXDR = data[i];

    }

}*/


void EEPROM_write_I2C(uint8_t devAddr,uint16_t memAddr, uint8_t* data, int size){



  I2C1->CR2 = 0;
  I2C1->CR2 |= (uint32_t)(devAddr << 1);
  I2C1->CR2 |= (uint32_t)((size + 2)<< 16);
  I2C1->CR2 |= (1U << 25); /*Auto-end*/
  I2C1->CR2 |= (1U << 13); /*Generate start*/

  while(!(I2C1->ISR & (1 << 1))); //high address
  I2C1->TXDR = (uint32_t)(memAddr >> 8);

  while(!(I2C1->ISR & (1 << 1))); //low address
  I2C1->TXDR = (uint32_t)(memAddr & 0xFF);

  while(size){
    while(!(I2C1->ISR & (1 << 1)));
    I2C1->TXDR = (*data++); /*DATA SEND*/
    size--;
  }
}

void EEPROM_read_I2C(uint8_t devAddr,uint16_t memAddr, uint8_t
*data, int size){

  I2C1->CR2 = 0;
  I2C1->CR2 |= (uint32_t)(devAddr << 1);
  I2C1->CR2 |= (2U << 16); //Number of bytes
  I2C1->CR2 |= (1U << 13); //Generate Start

  while(!(I2C1->ISR & (1 << 1)));//high address
  I2C1->TXDR = (uint32_t)(memAddr >> 8);


  while(!(I2C1->ISR & (1 << 1))); //low address
  I2C1->TXDR = (uint32_t)(memAddr & 0xFF);


  while(!(I2C1->ISR & (1 << 6))); //is transmission complete


  //read data
  I2C1->CR2 = 0;
  I2C1->CR2 |= (uint32_t)(devAddr << 1);
  I2C1->CR2 |= (1U << 10); //Read mode
  I2C1->CR2 |= (uint32_t)(size << 16); //Number of bytes
  I2C1->CR2 |= (1U << 25); //AUTOEND
  I2C1->CR2 |= (1U << 13); //Generate start


  while(size){
    while(!(I2C1->ISR & (1 << 2)));
    (*data++) = (uint8_t)I2C1->RXDR;
    size--;
  }
}


/*void multi_Read_I2C(uint8_t devAddr, uint8_t regAddr, uint8_t
*data, uint32_t num){
  //Write operation (Send address and register to read)
  I2C1->CR2 = 0; //reset control reg2
  I2C1->CR2 |= ((uint32_t)devAddr << 1);//slave address
  I2C1->CR2 |= (1U << 16); //Number of bytes
  I2C1->CR2 |= (1U << 13); //Generate Start
  while(!(I2C1->ISR & (1U << 1))); //TXIS
  I2C1->TXDR = (uint32_t)regAddr;

while(!(I2C1->ISR & (1U << 6))); //Transmission complete

  //Read operation (read data)
  I2C1->CR2 = 0;
  I2C1->CR2 |= ((uint32_t)devAddr << 1);
  I2C1->CR2 |= (1U << 10); //Read mode
  I2C1->CR2 |= (num << 16); //Number of bytes
  I2C1->CR2 |= (1U << 15); //NACK=Not acknowledge
  I2C1->CR2 |= (1U << 25); //Autoend
  I2C1->CR2 |= (1U << 13); //Generate Start

  for(int i=0 ; i<num; i++){
    while(!(I2C1->ISR & (1U << 2)));//wait until RXNE=1
    data[i] = (uint8_t)I2C1->RXDR;
  }
}
*/

void GPIO_Config(void){
  RCC->IOPENR |= (1U << 0); //Enable clock for GPIOA
  RCC->APBENR1 |= (1U << 17); //Enable clock for USART2
  GPIOA->MODER &= ~(3U << 2*2);
  GPIOA->MODER |= (2U << 2*2);
  GPIOA->AFR[0] &= ~(0xFU << 4*2);
  GPIOA->AFR[0] |= (1 << 4*2);
  GPIOA->MODER &= ~(0xFU << 2*3);
  GPIOA->MODER |= (2U << 2*3);
  GPIOA->AFR[0] &= ~(0xFU << 4*3);
  GPIOA->AFR[0] |= (1 << 4*3);
}

void print(char *buf){
  int len = 0;
  while(buf[len++] != '\0');
  _write(0, buf, len);
}

void printChar(uint8_t c){
  USART2->TDR = (uint16_t) c;
  while(!(USART2->ISR & (1 << 6))); // 6.bit transmission complete
}

int _write(int fd, char *ptr, int len) {
  (void)fd;
  for (int i=0; i<len; ++i){
    printChar(ptr[i]);
  }
  return len;
}

void USART_Config(uint16_t baud){
  USART2->CR1 = 0;
  USART2->CR1 |= (1U << 2); //USART1 receiver enable
  USART2->CR1 |= (1U << 3); //USART1 transmitter enable
  USART2->CR1 |= (1U << 5); //RX Interrupt enable
  USART2->BRR = (uint16_t)(SystemCoreClock / baud); //Setting
  USART2->CR1 |= (1U << 0); //USART2 enable
  NVIC_SetPriority(USART2_IRQn , 1);
  NVIC_EnableIRQ(USART2_IRQn);
}

void delay(uint32_t s){
    for(;s>0;s--);
    }
```