GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC335

MICROPROCESSORS LAB

LAB 6 Report

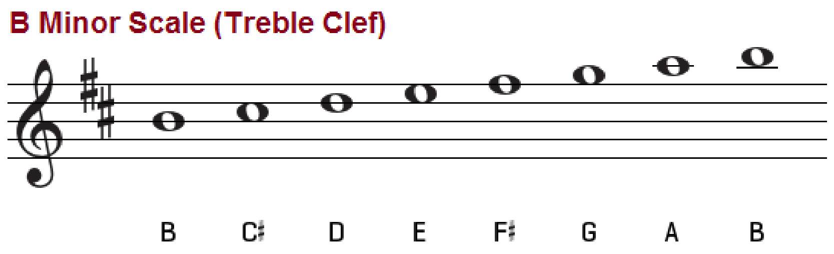
Prepared by
171024086 Berat KIZILARMUT

1. Introduction

In this Lab, we will be experimenting with Pulse Width Modulation operations in problem1. I will be connecting a 4x4 keypad which will have 15 notes and 1 button for rest on it. When a button for a note is pressed that said note will sustain indefinitely. On problem2 I will be connecting an IMU sensor via I2C and UART.

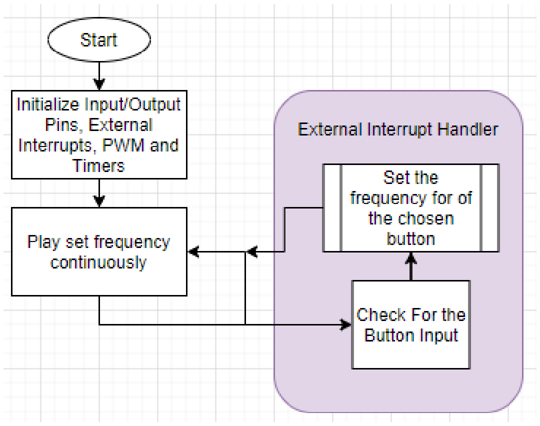
2. Problem 1

In this problem I will be designing a basic synthesizer. I will be connecting a 4-digit seven segment display, a 4x4 keypad and a sound output circuit including capacitors, resistors, potentiometer, amplifier and a speaker. 15 of the 16 buttons on the keypad will each get their own tone and will play the said tone indefinitely. Remaining button will be the rest button to silence the output.

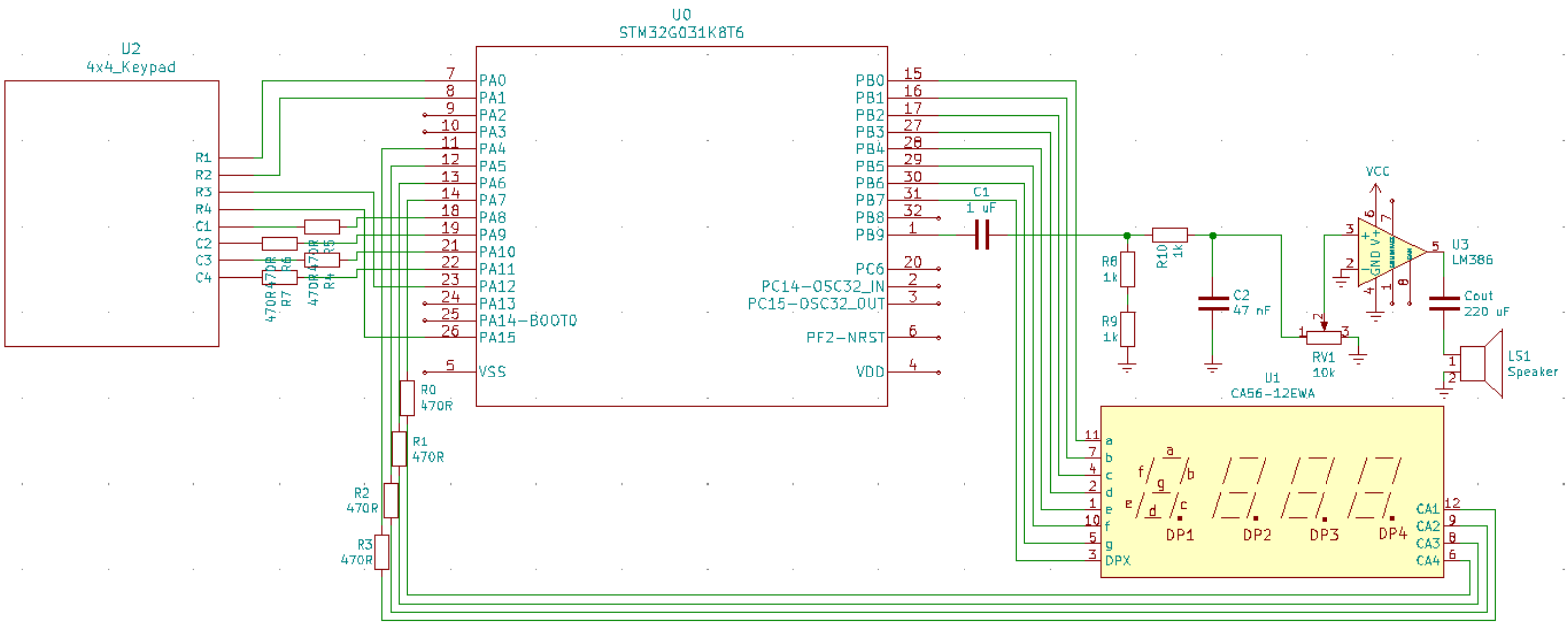


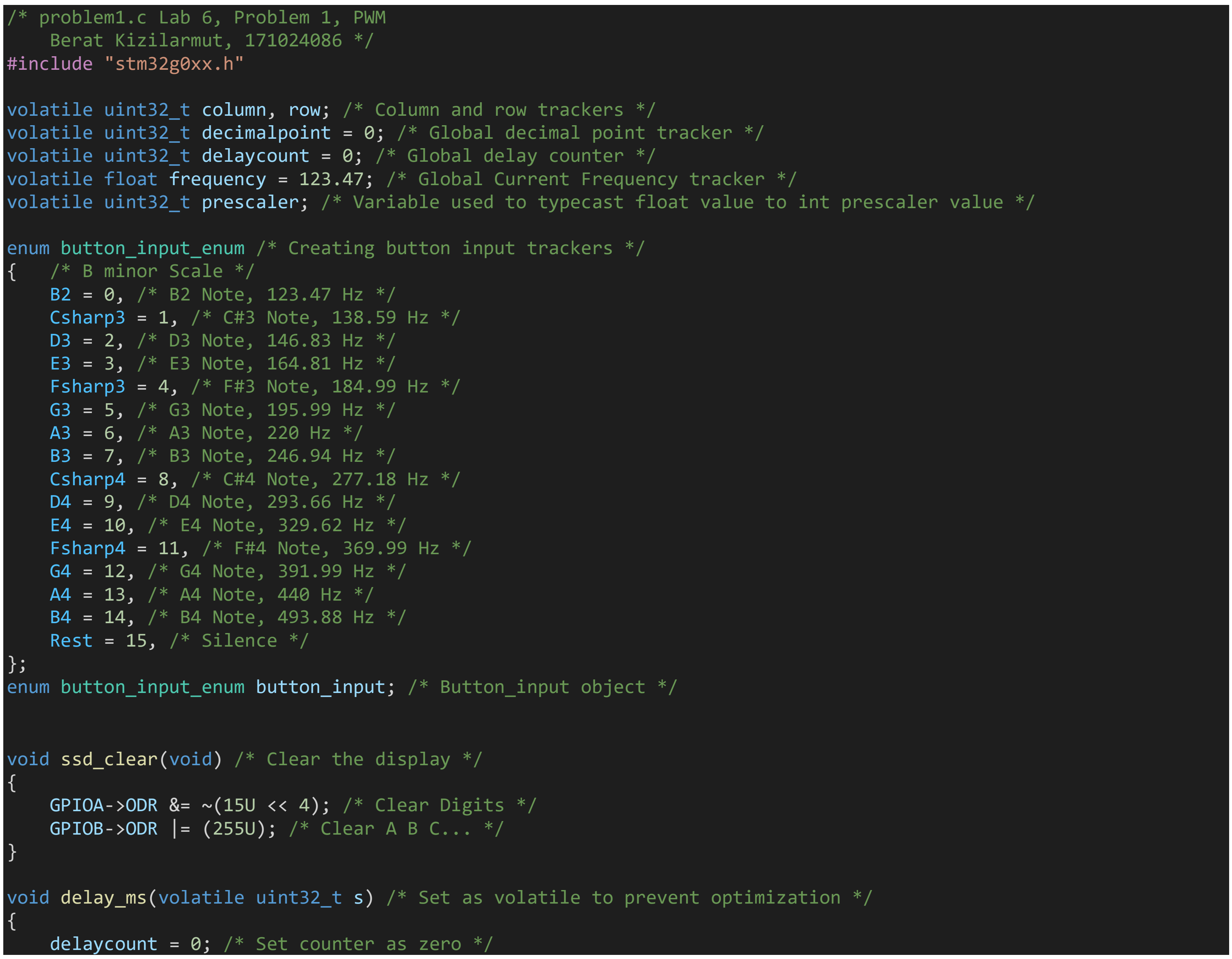
I will be using B minor scale for 2 octaves starting from B2 note. B minor scale is a series of notes as follows, B, C#, D, E, F#, G, A and B an octave higher. Two scales back to back have one B note overlapping so I will have a spare button on the keypad. This button will be the silence/rest button. My notes will be ranging from 123.47 Hz, B2 note, to 493.88 Hz, B4 note.

Explaining the code side; I’ve created an initialization function that sets up the board. In this setup process 4x4 keypad is connected via setting pins PA8 to PA11 as columns of the keypad and pins PA0 PA1 PA12 and PA15 are connected as rows of the keypad. External interrupt is set for the keypad. After the keypad setup, seven segment display is setup utilizing pins PB0 to PB7 as A B C D E F G and decimal points pins of the SSD. Pins PA4 to PA7 are setup as D1 D2 D3 D4 pins of the SSD. After that general-purpose time TIM1 is setup to use as a delay tool. Prescaler is set to 16000 to use the delay numbers as milliseconds. Timer17 is set up as PWM on PB9. Firstly, enabled the TIM17 clock, set the correct alternate functions and operating parameters.On the external interrupt handler I check for the 4x4 keypad input using the function I’ve created called “keypad_check” which detects the exact button pressed by toggling rows and columns and sets the frequency and button_input values correctly depending on the pressed button. I’ve created a SSD print function that prints the float frequency value of the currently playing tone or prints “rEST” if the rest button is pressed.


Start
Initialize Input/Output Pins, External Interrupts, PWM and Timers
Play set frequency continuously
External Interrupt Handler
Set the frequency for of the chosen button
Check For the Button Input

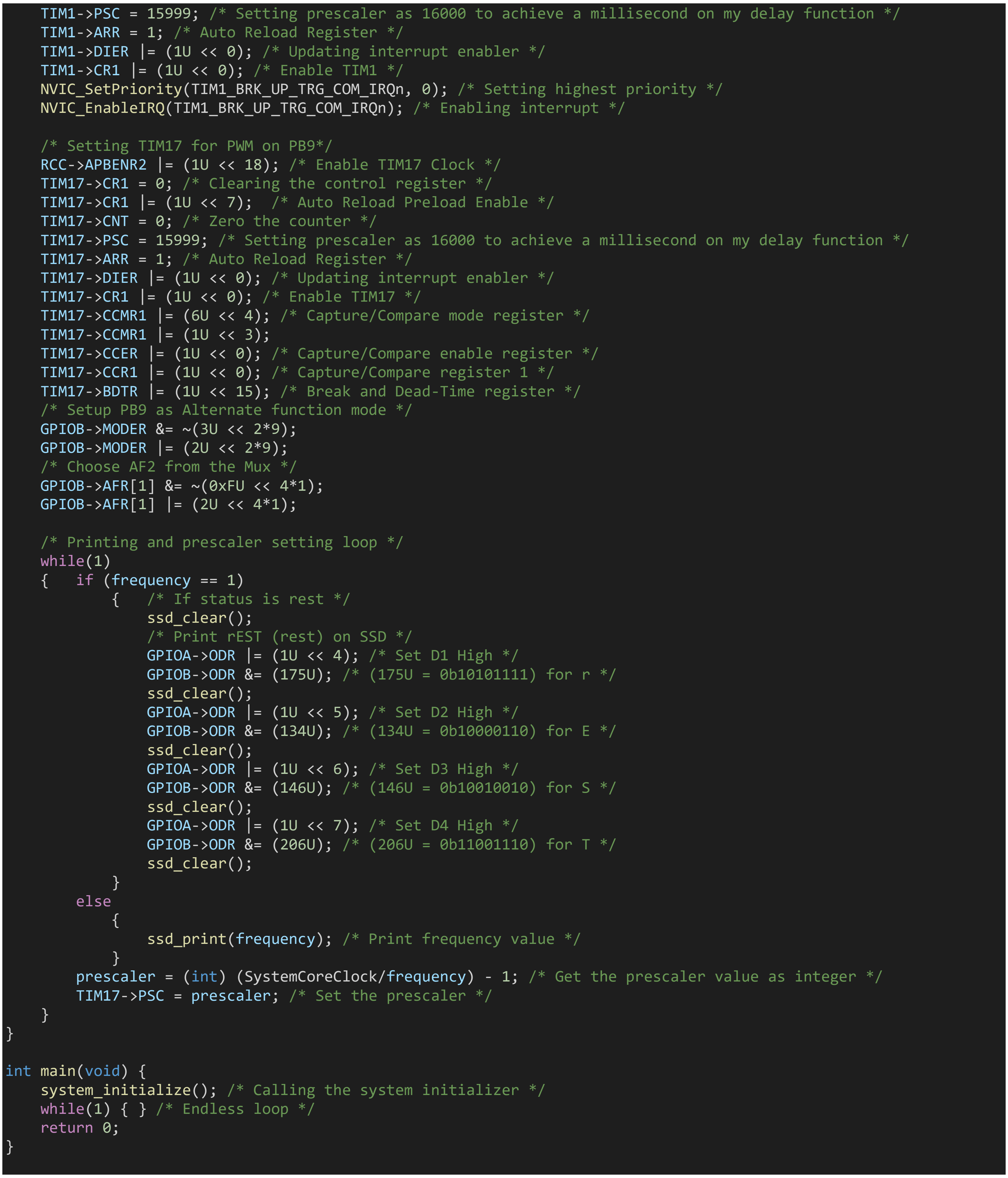
On the hardware side; I could not get the microprocessor create a sound I am having troubles detecting my core status. Keypad and SSD works as expected. I’ve reset the core several times, but I can’t seem to solve the problem. I’ve been connecting the speaker via crocodile jumper cables.





```c
/* problem1.c Lab 6, Problem 1, PWM
    Berat Kizilarmut, 171024086 */
#include "stm32g0xx.h"

volatile uint32_t column, row; /* Column and row trackers */
volatile uint32_t decimalpoint = 0; /* Global decimal point tracker */
volatile uint32_t delaycount = 0; /* Global delay counter */
volatile float frequency = 123.47; /* Global Current Frequency tracker */
volatile uint32_t prescaler; /* Variable used to typecast float value to int prescaler value */

enum button_input_enum /* Creating button input trackers */
{   /* B minor Scale */
    B2 = 0, /* B2 Note, 123.47 Hz */
    Csharp3 = 1, /* C#3 Note, 138.59 Hz */
    D3 = 2, /* D3 Note, 146.83 Hz */
    E3 = 3, /* E3 Note, 164.81 Hz */
    Fsharp3 = 4, /* F#3 Note, 184.99 Hz */
    G3 = 5, /* G3 Note, 195.99 Hz */
    A3 = 6, /* A3 Note, 220 Hz */
    B3 = 7, /* B3 Note, 246.94 Hz */
    Csharp4 = 8, /* C#4 Note, 277.18 Hz */
    D4 = 9, /* D4 Note, 293.66 Hz */
    E4 = 10, /* E4 Note, 329.62 Hz */
    Fsharp4 = 11, /* F#4 Note, 369.99 Hz */
    G4 = 12, /* G4 Note, 391.99 Hz */
    A4 = 13, /* A4 Note, 440 Hz */
    B4 = 14, /* B4 Note, 493.88 Hz */
    Rest = 15, /* Silence */
};
enum button_input_enum button_input; /* Button_input object */


void ssd_clear(void) /* Clear the display */
{
    GPIOA->ODR &= ~(15U << 4); /* Clear Digits */
    GPIOB->ODR |= (255U); /* Clear A B C... */
}

void delay_ms(volatile uint32_t s) /* Set as volatile to prevent optimization */
{
    delaycount = 0; /* Set counter as zero */
    while(1) /* Continuously checks the counter */
    {
        if(delaycount == s)
        return;
    }
}

float float_disassembly(float floatnumber)
{
    if(floatnumber >= 1000)
    {
        decimalpoint=0;
        floatnumber = floatnumber; /* Do nothing, it's already filled up to 4 digits */
    }
    else if (floatnumber >= 100)
    {
        decimalpoint=3; /* Put the decimal point to the third position from left */
        floatnumber = (10 * floatnumber);
    }
    else if (floatnumber >= 10)
    {
        decimalpoint=2; /* Put the decimal point to the second position from left */
        floatnumber = (100 * floatnumber);
    }
    else
    {
        decimalpoint=1; /* Put the decimal point to the first position from left */
        floatnumber = (1000 * floatnumber);
    }
    return floatnumber;
}

void ssd_printdigit(uint32_t s) /* Prints the values depending on the number */
{
    switch(s)
    {
        case 0: /* Printing 0 */
            GPIOB->ODR &= (0xC0U);
            break;

        case 1: /* Printing 1 */
            GPIOB->ODR &= (0xF9U);
            break;

        case 2: /* Printing 2 */
            GPIOB->ODR &= (0xA4U);
            break;

        case 3: /* Printing 3 */
            GPIOB->ODR &= (0xB0U);
            break;

        case 4: /* Printing 4 */
            GPIOB->ODR &= (0x99U);
            break;

        case 5: /* Printing 5 */
            GPIOB->ODR &= (0x92U);
            break;

        case 6: /* Printing 6 */
            GPIOB->ODR &= (0x82U);
            break;

        case 7: /* Printing 7 */
            GPIOB->ODR &= (0xF8U);
            break;

        case 8: /* Printing 8 */
            GPIOB->ODR &= (0x80U);
            break;

        case 9: /* Printing 9 */
            GPIOB->ODR &= (0x90U);
            break;
    }
}

void ssd_print(float value)
{
    value=float_disassembly(value); /* Disassemble the input to get decimal point */
    /* Gather the digit by digit values using integer rounding */
    uint32_t d1=0, d2=0, d3=0, d4=0;
    d1=value/1000;
    value=value-(d1*1000);
    d2=value/100;
    value=value-(d2*100);
    d3=value/10;
    value=value-(d3*10);
    d4=value;
    ssd_clear();
    /* Print Thousands Digit */
    GPIOA->ODR |= (1U << 4); /* Set D1 High */
    ssd_printdigit(d1); /* Print Value to D1 */
    if(decimalpoint == 1) /* Put the decimal point if requested */
    GPIOB->ODR &= ~(1U << 7);
    ssd_clear();
    /* Print Hundreds Digit */
    GPIOA->ODR |= (1U << 5); /* Set D2 High */
    ssd_printdigit(d2); /* Print Value to D2 */
    if(decimalpoint == 2) /* Put the decimal point if requested */
    GPIOB->ODR &= ~(1U << 7);
    ssd_clear();
    /* Print Tens Digit */
    GPIOA->ODR |= (1U << 6); /* Set D3 High */
    ssd_printdigit(d3); /* Print Value to D3 */
    if(decimalpoint == 3) /* Put the decimal point if requested */
    GPIOB->ODR &= ~(1U << 7);
    ssd_clear();
    /* Print Ones Digit */
    GPIOA->ODR |= (1U << 7); /* Set D4 High */
    ssd_printdigit(d4); /* Print Value to D4 */
    ssd_clear();
}

void clear_rows(void)
{
    /* Setting all the rows as low */
    GPIOA->ODR &= ~(36867U); /* ~(36867U) = ~(1001000000000011) */
}

void activate_rows(void)
{
    /* Setting all the rows as high */
    GPIOA->ODR |= (36867U); /* 36867U = 1001000000000011 */
}

void keypad_check(void)
{   /* This function checks for the button row and column */
    /* Setting Row 1 (PA0) as 1, others as zero to check */
    clear_rows();
    GPIOA->ODR |= (1U);
    if((GPIOA->IDR >> 8) & (1U)) /* Checking CR1, PA8 */
    { /* C1 and R1 means button "1" */
        column = 1;
        row = 1;
        frequency = 123.47;
        button_input = B2;
    }

    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R1 means button "2" */
        column = 2;
        row = 1;
        frequency = 138.59;
        button_input = Csharp3;
    }

    if((GPIOA->IDR >> 10) & (1U)) /* Checking CR3, PA10 */
    { /* C3 and R1 means button "3" */
        column = 3;
        row = 1;
        frequency = 146.83;
        button_input = D3;
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R1 means button "A" */
        column = 4;
        row = 1;
        frequency = 164.81;
        button_input = E3;
    }

    /* Setting Row 2 (PA1) as 1, others as zero to check */
    clear_rows();
    GPIOA->ODR |= (1U << 1);
    if((GPIOA->IDR >> 8) & (1U)) /* Checking CR1, PA8 */
    { /* C1 and R2 means button "4" */
        column = 1;
        row = 2;
        frequency = 184.99;
        button_input = Fsharp3;
    }

    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R2 means button "5" */
        column = 2;
        row = 2;
        frequency = 195.99;
        button_input = G3;
    }

    if((GPIOA->IDR >> 10) & (1U)) /* Checking CR3, PA10 */
    { /* C3 and R2 means button "6" */
        column = 3;
        row = 2;
        frequency = 220;
        button_input = A3;
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R2 means button "B" */
        column = 4;
        row = 2;
        frequency = 246.94;
        button_input = B3;
    }
    /* Setting Row 3 (PA12) as 1, others as zero to check */
    clear_rows();
    GPIOA->ODR |= (1U << 12);
    if((GPIOA->IDR >> 8) & (1U)) /* Checking CR1, PA8 */
    { /* C1 and R3 means button "7" */
        column = 1;
        row = 3;
        frequency = 277.18;
        button_input = Csharp4;
    }

    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R3 means button "8" */
        column = 2;
        row = 3;
        frequency = 293.66;
        button_input = D4;
    }

    if((GPIOA->IDR >> 10) & (1U)) /* Checking CR3, PA10 */
    { /* C3 and R3 means button "9" */
        column = 3;
        row = 3;
        frequency = 329.62;
        button_input = E4;
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R3 means button "C" */
        column = 4;
        row = 3;
        frequency = 369.99;
        button_input = Fsharp4;
    }


    /* Setting Row 4 (PA15) as 1, others as zero to check */
    clear_rows();
    GPIOA->ODR |= (1U << 15);
    if((GPIOA->IDR >> 8) & (1U)) /* Checking CR1, PA8 */
    { /* C1 and R4 means button "*" */
        column = 1;
        row = 4;
        frequency = 391.99;
        button_input = G4;
    }

    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R4 means button "0" */
        column = 2;
        row = 4;
        frequency = 440;
        button_input = A4;
    }

    if((GPIOA->IDR >> 10) & (1U)) /* Checking CR3, PA10 */
    { /* C3 and R4 means button "#" */
        column = 3;
        row = 4;
        frequency = 493.88;
        button_input = B4;
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R4 means button "D" */
```

```c
        column = 4;
        row = 4;
        frequency = 1;
        button_input = Rest;
    }
    activate_rows(); /* Setting the rows high before leaving the check function*/
}

void EXTI4_15_IRQHandler (void) {
    delaycount = 0; /* Setting up for timeout */
    keypad_check(); /* Get button information */
    delay_ms(500); /* Delay so function does not run continiously */
    EXTI->RPR1 |= (15U << 8); /* Clearing pending PA8 to PA11 */
}

void TIM1_BRK_UP_TRG_COM_IRQHandler (void) /* Configuring TIM1 */
{
    delaycount++;
    TIM1->SR &= ~(1U << 0); /* Clear pending status */
}

void system_initialize(void) /* Setting up the board and subsystems */
{
    /* Enable GPIOA and GPIOB clock */
    RCC->IOPENR |= (3U); /* 3U = 11 */
    /* Setting up the 4x4 Keypad */
    /* Setting up the external button interrupts */
    /* Setup PA8 to PA11 as input to use as columns on the keypad */
    GPIOA->MODER &= ~(255U << 8*2); /* ~(255U) = ~(11111111) */
    GPIOA->PUPDR &= ~(85U << 8*2); /* ~(85U) = ~(01010101) */
    GPIOA->PUPDR |= (170U << 8*2); /* 170U = 10101010 */
    /* Setting up the interrupt operation for columns */
    EXTI->RTSR1 |= (15U << 8); /* Setting the trigger as rising edge */
    EXTI->EXTICR[2] &= ~(1U << 8*0); /* Setting for PA8 */
    EXTI->EXTICR[2] &= ~(1U << 8*1); /* Setting for PA9 */
    EXTI->EXTICR[2] &= ~(1U << 8*2); /* Setting for PA10 */
    EXTI->EXTICR[2] &= ~(1U << 8*3); /* Setting for PA11 */
    EXTI->IMR1 |= (15U << 8); /* Interrupt mask register */
    EXTI->RPR1 |= (15U << 8); /* Rising edge pending register, Clearing pending PA8 to PA11 */
    NVIC_SetPriority(EXTI4_15_IRQn, 1); /* Setting priority */
    NVIC_EnableIRQ(EXTI4_15_IRQn); /* Enabling the interrupt function */
    /* Setting up the pins PA0, PA1, PA12 and PA15 as output to use a rows on the keypad */
    GPIOA->MODER &= ~(3U << 0*2); /* Zero PA0 */
    GPIOA->MODER |= (1U << 0*2); /* Set first bit one on PA0 */
    GPIOA->MODER &= ~(3U << 1*2); /* Zero PA1 */
    GPIOA->MODER |= (1U << 1*2); /* Set first bit one on PA1 */
    GPIOA->MODER &= ~(3U << 12*2); /* Zero PA12 */
    GPIOA->MODER |= (1U << 12*2); /* Set first bit one on PA12 */
    GPIOA->MODER &= ~(3U << 15*2); /* Zero PA15 */
    GPIOA->MODER |= (1U << 15*2); /* Set first bit one on PA15 */
    /* Have to set the rows as high to detect an initial input */
    GPIOA->ODR |= (36867U); /* 36867U = 1001000000000011 */

    /* Setting up the Seven Segment Display*/
    /* Set PB0 to PB7 as output to use as A B C D E F G and decimal point pins of the SSD */
    GPIOB->MODER &= ~(65535U); /* Setting first 16 bits as zero */
    GPIOB->MODER |= (21845U); /* Setting odd bits as one */
    /* Set PA4 to PA7 as output to use as D1 D2 D3 D4 pins of the SSD */
    GPIOA->MODER &= ~(65280U); /* Setting bits 8th to 16th as zero */
    GPIOA->MODER |= (21760U); /* Setting odd bits as one from 8th to 16th */

    /* Setting TIM1 for Delays*/
    RCC->APBENR2 |= (1U << 11); /* Enable TIM1 Clock */
    TIM1->CR1 = 0; /* Clearing the control register */
    TIM1->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM1->CNT = 0; /* Zero the counter */
```



```c
    TIM1->PSC = 15999; /* Setting prescaler as 16000 to achieve a millisecond on my delay function */
    TIM1->ARR = 1; /* Auto Reload Register */
    TIM1->DIER |= (1U << 0); /* Updating interrupt enabler */
    TIM1->CR1 |= (1U << 0); /* Enable TIM1 */
    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0); /* Setting highest priority */
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn); /* Enabling interrupt */

    /* Setting TIM17 for PWM on PB9*/
    RCC->APBENR2 |= (1U << 18); /* Enable TIM17 Clock */
    TIM17->CR1 = 0; /* Clearing the control register */
    TIM17->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM17->CNT = 0; /* Zero the counter */
    TIM17->PSC = 15999; /* Setting prescaler as 16000 to achieve a millisecond on my delay function */
    TIM17->ARR = 1; /* Auto Reload Register */
    TIM17->DIER |= (1U << 0); /* Updating interrupt enabler */
    TIM17->CR1 |= (1U << 0); /* Enable TIM17 */
    TIM17->CCMR1 |= (6U << 4); /* Capture/Compare mode register */
    TIM17->CCMR1 |= (1U << 3);
    TIM17->CCER |= (1U << 0); /* Capture/Compare enable register */
    TIM17->CCR1 |= (1U << 0); /* Capture/Compare register 1 */
    TIM17->BDTR |= (1U << 15); /* Break and Dead-Time register */
    /* Setup PB9 as Alternate function mode */
    GPIOB->MODER &= ~(3U << 2*9);
    GPIOB->MODER |= (2U << 2*9);
    /* Choose AF2 from the Mux */
    GPIOB->AFR[1] &= ~(0xFU << 4*1);
    GPIOB->AFR[1] |= (2U << 4*1);

    /* Printing and prescaler setting loop */
    while(1)
    {   if (frequency == 1)
            {   /* If status is rest */
                ssd_clear();
                /* Print rEST (rest) on SSD */
                GPIOA->ODR |= (1U << 4); /* Set D1 High */
                GPIOB->ODR &= (175U); /* (175U = 0b10101111) for r */
                ssd_clear();
                GPIOA->ODR |= (1U << 5); /* Set D2 High */
                GPIOB->ODR &= (134U); /* (134U = 0b10000110) for E */
                ssd_clear();
                GPIOA->ODR |= (1U << 6); /* Set D3 High */
                GPIOB->ODR &= (146U); /* (146U = 0b10010010) for S */
                ssd_clear();
                GPIOA->ODR |= (1U << 7); /* Set D4 High */
                GPIOB->ODR &= (206U); /* (206U = 0b11001110) for T */
                ssd_clear();
            }
        else
            {
                ssd_print(frequency); /* Print frequency value */
            }
        prescaler = (int) (SystemCoreClock/frequency) - 1; /* Get the prescaler value as integer */
        TIM17->PSC = prescaler; /* Set the prescaler */
    }
}

int main(void) {
    system_initialize(); /* Calling the system initializer */
    while(1) { } /* Endless loop */
    return 0;
}
```

3. References

1. RM0444 Reference Manual, STM32G0x1 advanced Arm®-based 32-bit MCUs, ST Microelectronics
2. Piano Key Frequencies, Wikipedia, https://en.wikipedia.org/wiki/Piano_key_frequencies