GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC334

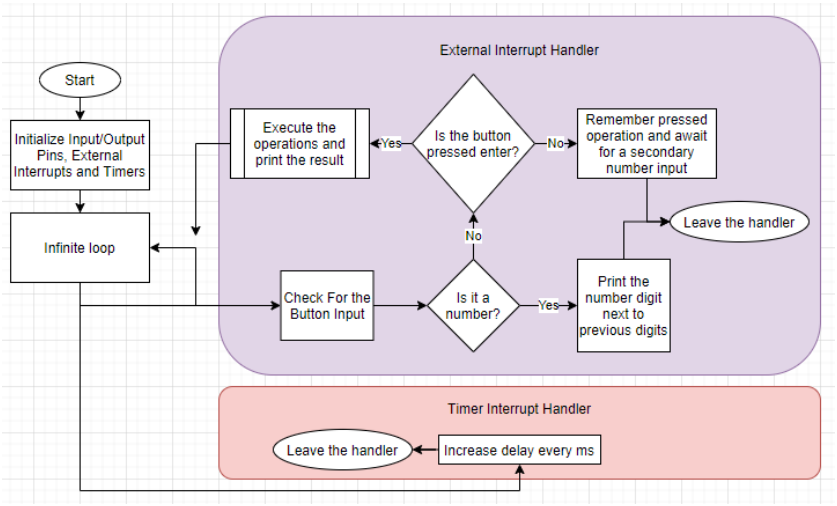
MICROPROCESSORS

Project 2 Report

Prepared by
171024086 Berat KIZILARMUT

1. Introduction

In this I will design a scientific calculator from the ground up without any additional pre-made libraries and functions (except math.h). This calculator will have addition, subtraction, multiplication, division, logarithmic, ln, square root, number squared and basic trigonometric functions. Calculations that have their results lay within 9999 to -999 will be displayed without an overflow. Input device will be a 4x4 Numpad, output device will be a 4-Digit Seven Segment Display. I will be providing flowcharts, hardware diagrams and photographs of the project.



2. Approach and Details Explained

Let's explain the limitations, functionalities and using methods of the calculator. When the board is powered up for the first-time display will show "1786" which is first 2 and last 2 digits of my student ID which is the IDLE state. Calculator will be waiting in idle state until a button press.

Inputs will be registered with a 4x4 numpad that has numbers from 0 to 9, letters from A to D and two symbols, * and #. Letters from A to D will be used as follows in order; addition, subtraction, multiplication and division. Star (*) Symbol will be used as sub menu for scientific mode functions. On this sub menu secondary button presses are expected, for example if we were to press A after pressing Star (*) we will be doing log operation. Sub menu functions are listed as, A for Log, B for Ln, C for Square Root, D for Number Squared and Star (*) for trigonometric functions sub menu. In the trigonometric functions sub menu, we expect yet another key press, pressing A will get sinus, B for cosines, C for tangent, D for cotangent and button E will give us the pi number as an input. For example, to use a cos operator we will be pressing Star (*) Star (*) and B in order. Trigonometric functions will be working with Degree as their input, not radian**.** Lastly, # symbol will be enter/equal button.

Outputs will be displayed on a 4-Digit Seven Segment Display. Numbers can be printed from -999 to 9999 since we only have 4 digits and we are not using shifting digit screens. Float numbers also will be printed but only up to 4 digits total, including integer parts. For example, we can print "12.34" but can't print "123.45", we will have to print "123.4". I have chosen to use pins PA4 to PA7 as my digit inputs for the SSD and pins PB0 to PB7 as my A B C D E F G and digit point pins.

3. Task List

Created this task list to list what I think I have to create to complete this project. However, what I actually need might be very different.

- Initialize function ✓: A function that setup the board, GPIOs, interrupts and watchdog.
- State machine X: A state machine that will track how many inputs for a number has been made, what state the calculator is in etc.
- Decimal conversion function ✓: A function that will transform the given decimal values to digit by digit values and float points
- SSD printing function ✓: A function that will display given digits in a non-flickering fashion.
- Button input interrupts ✓: A function that will detect and manage number inputs and menu inputs.
- Keypad input decipher function ✓: A function that will test and find what button is pressed, which row and which column.
- Mathematical functions ✓: A function that will do the mathematical operations depending on menu inputs

4. Explaining the Functions and Modules

In this section I will be explaining each of my functions, what I've tried to achieve and how I achieved that.

4.1 Initialization Function

This is the first function that we must call in the main. This function initializes everything. Firstly, I clock the GPIOA and GPIOB peripherals. After the peripherals I start setting the 4x4 Keypad up. As a side note I like to use binary to decimal converters to complete MODER and ODR operations. Rather than setting the ODR or MODER's by pin by pin, I write the desired pin order in binary and convert it to decimal and set all the pins with a single line wherever I can. I've set the GPIOA MODER and PUPDR register using such technique however It was much easier to set the EXTICR's of the pins independent. I've utilized the same SSD pins and layout I've used on prior labs which was PB to PB7 as A B C D E F G and decimal point pins of the SSD and PA4 to PA7 as D1 D2 D3 and D4 pins of the SSD. For the keypad I used PA8 to PA11 as input columns and set an interrupt routine to these pins. I used PA0 PA1 PA12 and PA15 as my keypad rows because there were no pins in order left on the board. Implemented a general-purpose timer, TIM1, to utilize in delay function. Timer interrupt priority is set as highest.

4.2 Printing Functions

I have created a three-part printing function. Firstly, there is ssd_print, which is the main printing function. This function takes a float input, checks if it's overflowing, if it does overflow it prints "ouFL" on the SSD. Next it checks the number if its negative. If the number is negative, number is manipulated in to being positive and manually "-" sign is printed on the digit 4. In both cases numbers go to a sub function, float_disassembly, which checks for many digits that the number has, manipulates it in to a writeable number and takes note on the supposed decimal point location. Then the number is sent to the digit printing function, ssd_printdigit, after setting the digit pins. After digit by digit printing operation, display is cleared with the ssd_clear function.

4.3 Keypad Functions and Interrupts

Created several sub functions to use the keypad. Firstly, because I've used pins PA8 to PA11 as column pins for the keypad, I used external interrupt handler EXTI4_15_IRQHandler.

When an input from the keypad is detected we arrive at the interrupt handler. First function called in the interrupt handler is keypad_check, which firstly sets every row voltage as zero using clear_rows function then sets a row high and checks for column input. Using this function exact button is determined and the states and trackers are set depending on the button, for example if we press button "3" since this button is a number, button_type is set as number. Rows voltages are set as high before leaving this function using activate_rows function.

After keypad input has been determined, two different switches await us depending on the button_type. Both of these switches check for the current and last state information and act accordingly. Number is printed and pending is cleared.

I had some problems detecting keypad presses true, I could not determine what causes this and could not solve this particular problem in time.

4.4 Timer and Delay Functions

Set-up a general-purpose timer, TIM1, on system initializer to use on the delay operations. Created a delay_ms function that delays for the given integer millisecond amount. This function utilizes the fact that the timer interrupt handler that I've set up continuously increases a global variable, delaycount, and since I've set the timer prescaler as 16000 this corresponds for a millisecond. When we call the function delay_ms, function resets the delaycount value to zero and checks continuously for the value to reach the required amount. Using timer, after no button presses for 10 seconds the calculator returns back to idling.

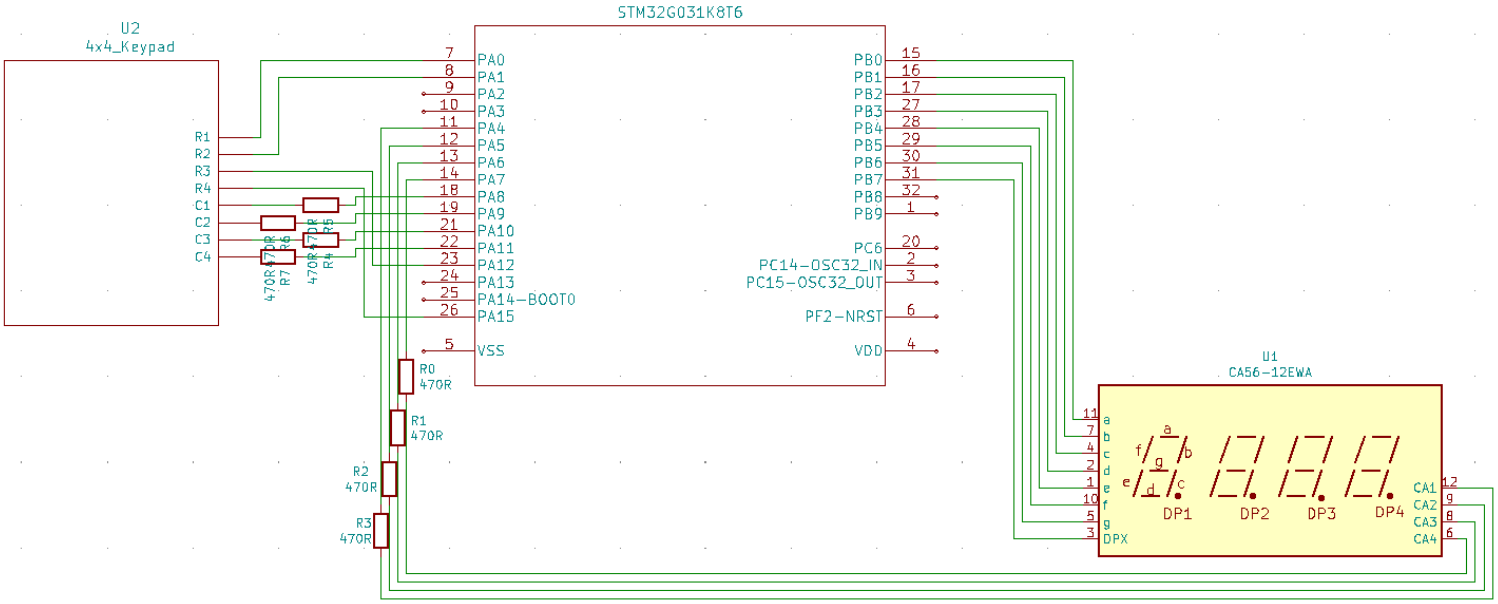
4.5 Other Smaller Functions

I've created dedicated error function that is called when something goes wrong. This function prints errors by checking the numbers, if they are overflowing it prints "ouFL", if they are not overflowing it assumes something else went wrong or some operations had some invalid operations occur and prints "Invd".

I created printing functions in the continuous while loop within main that display results or error messages for 5 seconds and take the program back to idling after 10 seconds with no action occur.

I created an operation executor function that executes previously gathered operation datas and number when result button is pressed. After this function is completed, result is printed for 5 seconds.

5. Hardware

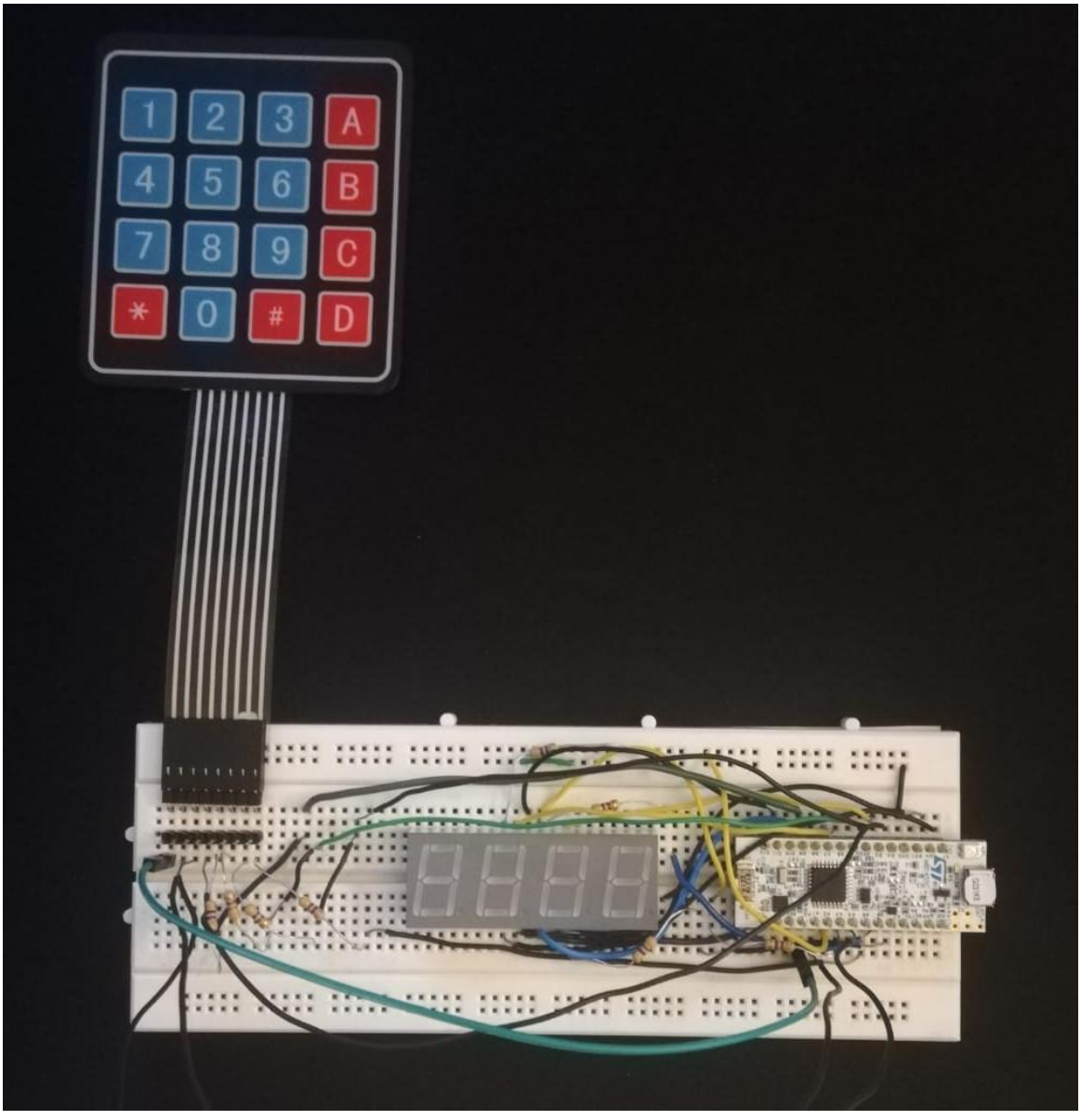


Breadboard is set by connecting 4-Digit Seven Segment Display and 4x4 Keypad to the breadboard and connecting the pre-determined pins. Totally 8 resistors, 470 ohms to be exact, are used to limit the currents flowing through the SSD and Keypad. Since on both of these components have some paths that connect with each other serially, I've used only one current limiting resistor per path. Resistors are connected to the column pins of the keypad and the digit pins of the SSD. While debugging and testing, I sometimes used a jumper to connect between column and row pins of the keypad rather than actually pressing the keypad because keypad inputs sometimes had some irregularities, however I am not sure if this is my fault or the components fault.

Total component and cost list given below;

#	Part Name	Link	Amount	Unit Price	Total Price
1	STM32 G031K8T Development Board	https://www.empastore.com/stm32-islemci-kiti-nucleo-g031k8	1	135.41 TL	135.41 TL
2	4-Digit Seven Segment Display Common Anode	https://www.direnc.net/14mm-4lu-anot-display	1	7.21 TL	7.21TL
3	470 Ohm Resistor	https://www.robotistan.com/14w-470r-direnc-paketi-10-adet	8	0.046 TL	0.37 TL
4	4x4 Keypad	https://www.direnc.net/4x4-mebram-tus-takimi	1	5.41 TL	5.41 TL
6	Breadboard	https://www.direnc.net/tekli-breadboard	1	8.56 TL	8.56 TL
7	Jumper Cable M-M	https://www.direnc.net/40-adet-erkek-erkek-jumper-20cm	20	0.093 TL	1.86 TL
				Total	158.82 TL

Breadboard layout picture given below;



6. Conclusion

In this project I've learned a lot about mcu state operations. I've learned how hard it becomes after several state layers to keep track of things and keep the code in working order. I have utilized most of what I've learned from previous labs and homework's and had to add new skills on top of that. I had to learn how a 4x4 keypad works with only 8 pins. On previous tasks I connected a resistor to almost every pin but now I've learned that one 470-ohm resistor per path is enough.

6.1 Problems, Failures and Improvement Ideas

I had many problems with my keypad. I used the guide I have attached on my reference page and connected the keypad exactly as it said. My keypad inputs were not reliable at all and some rows even refused to work at all sometimes. I seriously don't know if this is my own fault or the keypad itself is fault or something else went wrong but I get correct results with just shorting the column and row pins with jumpers.

I've failed to set the scientific function states correctly due to running out of time, they may or may not work. My keypad does not reliably work.

Improvement ideas that I have are; on keypad input detection function, even after a keypad press is correctly detected and column row data is in function still checks for every if statement. If I can make the function finish after getting the required information, I could save some more processing power. Another improvement idea is on state machine parts I could merge or chain some states and even some functions to save more processing power.

7. References

1. RM0444 Reference Manual, STM32G0x1 advanced Arm®-based 32-bit MCUs, ST Microelectronics
2. 4x4 Keypad Module, Components101, https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet

8. Code

```
/* main.c Probject 2, Scientific Calculator
    Berat Kizilarmut, 171024086 */
#include "stm32g0xx.h"
#include <math.h>
volatile float number1 = 1786, number2 = 0, newnumber, floatnumber; /* Creating the glob
al variables*/
volatile char input; /* Input return variable */
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 */

enum state_enum /* Setting the state tracker enum */
```

```
{
    idle = 0, first_input = 1, next_input = 2, scientific = 3, submenu = 4 , result = 5,
 error = 6, result_printing = 7
};
enum state_enum state; /* State object */
enum state_enum previous_state; /* Previous State object */

enum button_type_enum /* Creating button type trackers */
{
    number = 0, operation = 1
};
enum button_type_enum button_type; /* Button_type object */

enum operation_type_enum /* Creating operation type trackers */
{
    addition = 0, subtraction = 1, multiplication = 2, division = 3, logarithmic = 4, ln
 = 5,
    squareroot = 6, squared = 7, sinus = 8, cosinus = 9, tangent = 10, cotangent = 11, p
i = 12
};
enum  operation_type_enum operation_type; /* Operation_type 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)
{   /* Check if there is overflow */
    if(value > 9999 || value < -999)
    {
        ssd_clear();
        /* Print ouFL (Overflow) on SSD */
        GPIOA->ODR |= (1U << 4); /* Set D1 High */
        GPIOB->ODR &= (163U); /* (163U = 0b10100011) for o */
        ssd_clear();
        GPIOA->ODR |= (1U << 5); /* Set D2 High */
        GPIOB->ODR &= (227U); /* (227U = 0b11100011) for v */
        ssd_clear();
        GPIOA->ODR |= (1U << 6); /* Set D3 High */
        GPIOB->ODR &= (142U); /* (142U = 0b10001110) for F */
        ssd_clear();
        GPIOA->ODR |= (1U << 7); /* Set D4 High */
        GPIOB->ODR &= (199U); /* (199U = 0b11000111) for L */
        ssd_clear();
        state = error;
        delaycount=0;
    }
```

```
    /* Check if the value is negative */
    if(value < 0)
    {
        value= (-10) * value; /* Turn the value positive and multiply it by 10 */
        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 */
        GPIOB->ODR &= (191U); /* (191U = 0b10111111) for - sign*/
        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(d1); /* 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(d2); /* 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(d3); /* Print Value to D4 */
        ssd_clear();
    }

    else
    {
    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 error_function(void)
{
    if(number1 > 9999 || number1 < -999)
    {
        /* Print ouFL (Overflow) on SSD */
        GPIOA->ODR |= (1U << 4); /* Set D1 High */
        GPIOB->ODR &= (163U); /* (163U = 0b10100011) for o */
        ssd_clear();
        GPIOA->ODR |= (1U << 5); /* Set D2 High */
        GPIOB->ODR &= (227U); /* (227U = 0b11100011) for v */
        ssd_clear();
        GPIOA->ODR |= (1U << 6); /* Set D3 High */
        GPIOB->ODR &= (142U); /* (142U = 0b10001110) for F */
        ssd_clear();
        GPIOA->ODR |= (1U << 7); /* Set D4 High */
        GPIOB->ODR &= (199U); /* (199U = 0b11000111) for L */
        ssd_clear();
        state = error;
    }
    else
    {
        /* Print Invd (Invalid) on SSD */
        GPIOA->ODR |= (1U << 4); /* Set D1 High */
        GPIOB->ODR &= (207U); /* (207U = 0b11001111) for I */
        ssd_clear();
        GPIOA->ODR |= (1U << 5); /* Set D2 High */
        GPIOB->ODR &= (171U); /* (171U = 0b10101011) for n */
        ssd_clear();
        GPIOA->ODR |= (1U << 6); /* Set D3 High */
        GPIOB->ODR &= (227U); /* (227U = 0b11100011) for v */
        ssd_clear();
        GPIOA->ODR |= (1U << 7); /* Set D4 High */
        GPIOB->ODR &= (161U); /* (161U = 0b10100001) for d */
        ssd_clear();
        state = error;
    }
}

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 operation_execute(void)
{
    switch(operation_type)
    {
        case addition: /* Complete the addition operation and print */
```

```
            number1 = number2 + number1;
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case subtraction: /* Complete the subtraction operation and print */
            number1 = number2 - number1;
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case multiplication: /* Complete the multiplication operation and print */
            number1 = number2 * number1;
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case division: /* Complete the division operation and print */
            if(number2 == 0) /* Cannot divide by zero */
            {
                number1=0; /* Zero the number */
                delaycount = 0; /* Set delay */
                state = error; /* Set Error State */
                error_function(); /* Go to error */
            }
            number1 = number2 / number1;
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case logarithmic: /* Complete the logarithmic operation and print */
            number1 = log10(number2);
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case ln: /* Complete the ln operation and print */
            number1 = log(number2);
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case squareroot: /* Complete the square root operation and print */
            number1 = sqrt(number2);
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case squared: /* Complete the squared operation and print */
            number1= number2 * number2;
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case sinus: /* Complete the sinus operation and print */
            number1 = sin(number2);
            ssd_print(number1);
            state = result_printing;
```

```
            delaycount = 0;
            break;

        case cosinus: /* Complete the cosinus operation and print */
            number1 = cos(number2);
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case tangent: /* Complete the tangent operation and print */
            number1 = tan(number2);
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;

        case cotangent: /* Complete the cotangent operation and print */
            number1 = 1 / tan(number2);
            ssd_print(number1);
            state = result_printing;
            delaycount = 0;
            break;
    }
}

void keypad_check(void)
{   /* This function checks for the button row and column */
    /* Also informs about the type of button press, is it a number or an operation */

    /* 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;
        input = '1';
        newnumber = 1;
        button_type = number;
    }

    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R1 means button "2" */
        column = 2;
        row = 1;
        input = '2';
        newnumber = 2;
        button_type = number;
    }

    if((GPIOA->IDR >> 10) & (1U)) /* Checking CR3, PA10 */
    { /* C3 and R1 means button "3" */
        column = 3;
        row = 1;
        input = '3';
        newnumber = 3;
        button_type = number;
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R1 means button "A" */
        column = 4;
        row = 1;
        input = 'A';
        operation_type = addition;
```

```
        button_type = operation;
    }

    /* 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;
        input = '4';
        newnumber = 4;
        button_type = number;
    }

    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R2 means button "5" */
        column = 2;
        row = 2;
        input = '5';
        newnumber = 5;
        button_type = number;
    }

    if((GPIOA->IDR >> 10) & (1U)) /* Checking CR3, PA10 */
    { /* C3 and R2 means button "6" */
        column = 3;
        row = 2;
        input = '6';
        newnumber = 6;
        button_type = number;
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R2 means button "B" */
        column = 4;
        row = 2;
        input = 'B';
        operation_type = subtraction;
        button_type = operation;
    }

    /* 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;
        input = '7';
        newnumber = 7;
        button_type = number;
    }

    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R3 means button "8" */
        column = 2;
        row = 3;
        input = '8';
        newnumber = 8;
        button_type = number;
    }

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

```
        row = 3;
        input = '9';
        newnumber = 9;
        button_type = number;
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R3 means button "C" */
        column = 4;
        row = 3;
        input = 'C';
        operation_type = multiplication;
        button_type = operation;
    }


    /* 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;
        input = '*';
        state = scientific;
        button_type = operation;
    }

    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R4 means button "0" */
        column = 2;
        row = 4;
        input = '0';
        newnumber = 0;
        button_type = number;
    }

    if((GPIOA->IDR >> 10) & (1U)) /* Checking CR3, PA10 */
    { /* C3 and R4 means button "#" */
        column = 3;
        row = 4;
        input = '#';
        state = result;
        button_type = operation;
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R4 means button "D" */
        column = 4;
        row = 4;
        input = 'D';
        operation_type = division;
        button_type = operation;
    }
    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 */
    /* Check for button type */
    if (button_type == number) /* Pressed button is a number */
    {
        switch(state)
        {
            case idle: /* If the last state was idle, we have to reset the number1 value
```

```
 */
                state=first_input; /* Since button type is number, this is first number
input state */
                previous_state = idle; /* Setting Previous State */
                number1 = newnumber; /* Carry the input to the number */
                ssd_print(number1); /* Print the current number */
                break;

            case first_input:
                if(previous_state == scientific || previous_state == submenu)
                {
                    state = next_input; /* Setting State */
                    number2 = number1; /* Carry the number to secondary number */
                    number1 = newnumber; /* Carry the input to the number */
                    ssd_print(number1); /* Print the current number */
                }
                else
                {
                    previous_state = first_input; /* Setting Previous State */
                    number1 = (10 * number1) + newnumber; /* Shift and carry the input t
o the number */
                    ssd_print(number1); /* Print the current number */
                }
                break;

            case next_input:
                if(previous_state == next_input)
                {
                    number1 = (10 * number1) + newnumber; /* Shift and carry the input t
o the number */
                    ssd_print(number1); /* Print the current number */
                }
                else
                {
                   /* number2 = number1; /* Switching values */
                    state = next_input; /* Setting State */
                    number1 = newnumber; /* Carry the input to the number */
                    previous_state = next_input; /* Setting Previous State */
                    ssd_print(number1); /* Print the current number */
                }
                break;
        }
    }

    if (button_type == operation) /* Pressed button is an operation */
    {
        switch(state)
        {
            case idle:
                number1=0; /* Zeroing number */
                state=next_input; /* Setting State */
                previous_state = idle; /* Setting Previous State */
                ssd_print(number1); /* Print the current number */
                break;

            case first_input:
                state=next_input; /* Setting State */
                previous_state = first_input; /* Setting Previous State */
                number2=number1; /* Switching values */
                number1=0; /* Zeroing number to get new inputs */
                ssd_print(number1); /* Print the current number */
                break;

            case next_input:
                number2=number1; /* Switching values */
                number1=0; /* Zeroing number to get new inputs */
```

```
                previous_state = first_input; /* Setting Previous State */
                state = first_input; /* Switching state to get new input */
                ssd_print(number1); /* Print the current number */
                break;

            case scientific:
                previous_state = scientific; /* Setting Previous State */
                if(column == 4 && row == 1)
                    operation_type = logarithmic; /* Setting operation type */
                if(column == 4 && row == 2)
                    operation_type = ln; /* Setting operation type */
                if(column == 4 && row == 3)
                    operation_type = squareroot; /* Setting operation type */
                if(column == 4 && row == 4)
                    operation_type = squared; /* Setting operation type */
                if(column == 1 && row == 4)
                    state = submenu;
                else
                {   /* Something went wrong */
                    number1=0; /* Zero the number */
                    delaycount = 0; /* Set delay */
                    state = error; /* Set Error State */
                    error_function(); /* Go to error */
                }
                break;

            case submenu:
                state = next_input; /* Setting State */
                previous_state = submenu; /* Setting Previous State */
                if(column == 4 && row == 1)
                    operation_type = sinus; /* Setting operation type */
                if(column == 4 && row == 2)
                    operation_type = cosinus; /* Setting operation type */
                if(column == 4 && row == 3)
                    operation_type = tangent; /* Setting operation type */
                if(column == 4 && row == 4)
                    operation_type = cotangent; /* Setting operation type */
                if(column == 1 && row == 4)
                    number1 = 3.14 ; /* Setting number as pi */
                else
                {   /* Something went wrong */
                    number1=0; /* Zero the number */
                    delaycount = 0; /* Set delay */
                    state = error; /* Set Error State */
                    error_function(); /* Go to error */
                }
                break;

            case result:
                state = result_printing; /* Setting State */
                previous_state = result; /* Setting Previous State */
                operation_execute(); /* Execute the operations and print result */
                break;
        }
    }
    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 SS
D */
    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 */
    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 */
    TIM1-
>PSC = 15999; /* Setting prescaler as 16000 to achieve a millisecond on my delay functio
n */
    TIM1->ARR = 1;
    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 */
}

int main(void) {
    system_initialize(); /* Calling the system initializer */
    while(1) {
        if(state == idle)
        { /* If state is idle set my school number and keep printing it */
            number1 = 1786;
            ssd_print(number1);
        }
        else if (state == error)
```

```
        { /* If state is error, keep printing error message for 5 seconds */
            if(delaycount==5000)
            {
                state=idle;
            }
            error_function();
        }
        else if (state == result_printing)
        { /* If state is result printing, keep printing result for 5 seconds */
            if(delaycount==5000)
            {
                state=idle;
            }
            ssd_print(number1);
        }
        else{
            if(delaycount==10000) /* If nothing happens for 10 seconds, go back to idlin
g */
            {
                state=idle;
            }
            ssd_print(number1);
        }
    } /* Endless loop */
    return 0;
}
```