GEBZE TECHNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC334

MICROPROCESSORS

Fully Operational Scientific Calculator

Project 2 Report

Prepared by
1801022022 – Alperen Karataş

1. Introduction

In this project, a fully operational scientific calculator has been designed. This calculator is a scientific calculator that can perform simple four transaction calculations, as well as trigonometric and scientific calculations. A 4x4 keypad was used for dialing, while a 4-digit SSD was used to see the numbers.

2. Theoretical Research

Before starting the project, knowing that information about the 4-digit seven segment display. Common-anode, common-cathode structures were learned for SSD and connections and pin inputs and outputs of common anode SSD structure to be used in the project were learned.

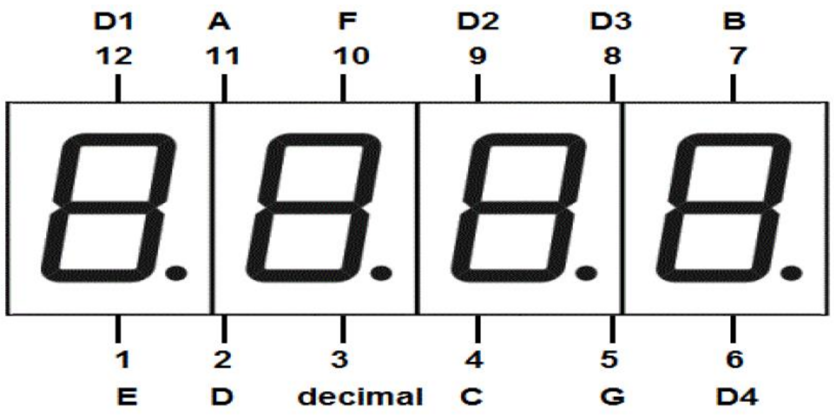


Figure 1. 4-digit Seven Segment Display

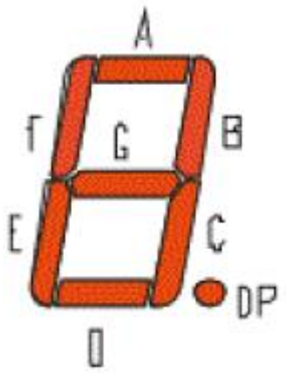


Figure 2. 1-digit Seven Segment Display

Figure 1 shows connections to the SSD used in the project. Figure 2 shows the locations of LED connections for A-B-C-D-E-F-G and DP (decimal point) for one digit.

In this project, the 4x4 keypad operating logic for dialing operations in the calculator is also examined.

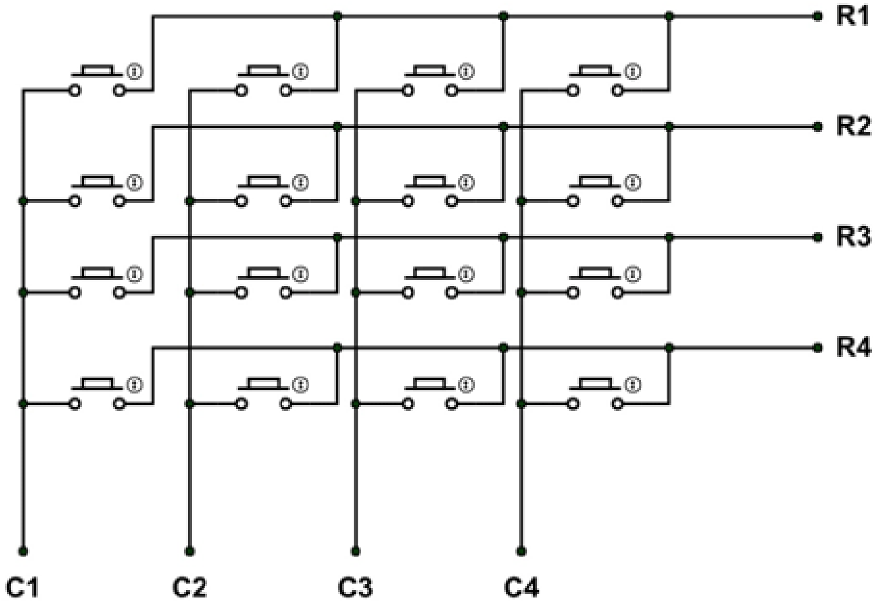


Figure 3. 4x4 keypad interior structure

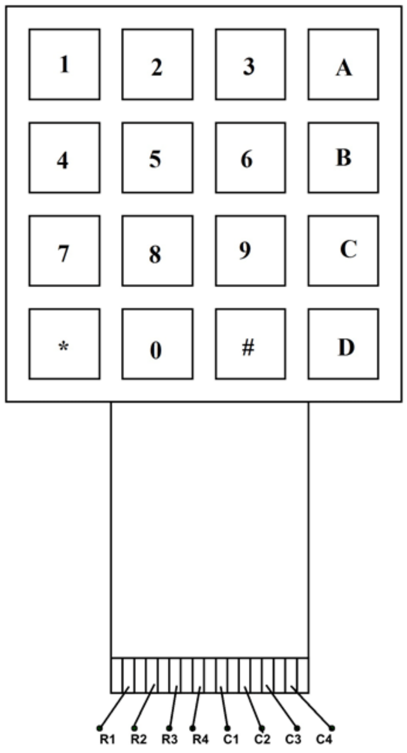


Figure 4. 4x4 keypad exterior structure

Figure 3 and Figure 4 have a keypad interior structure and exterior structure. Keypad works according to the logic of short-circuiting the part where the pressed button is located and understanding which button is pressed in the micro-processor with the flow of the current.

Keypad consists of row and column pins. When Figure 3 is examined, R1, R2, R3, and R4 express rows, while C1, C2, C3, and C4 express columns. When the button is pressed, the current will flow to the pin where the relevant column is located, depending on if the relevant row pin is 5V or 0V, and from there it will be possible to determine which button is pressed by flowing to the microprocessor.

3. Project Setup:

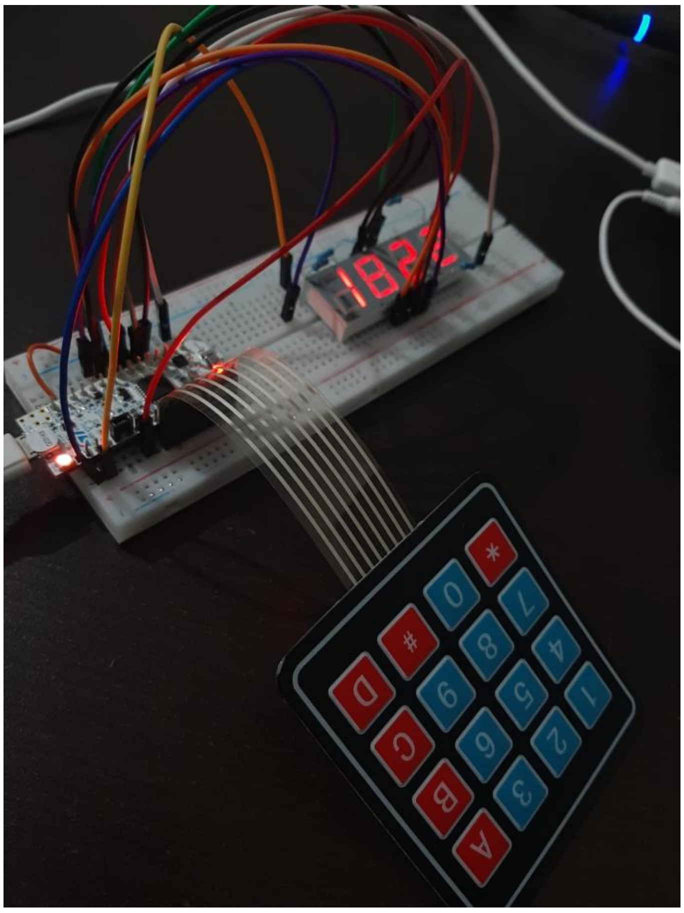


Figure 5. Project setup photo

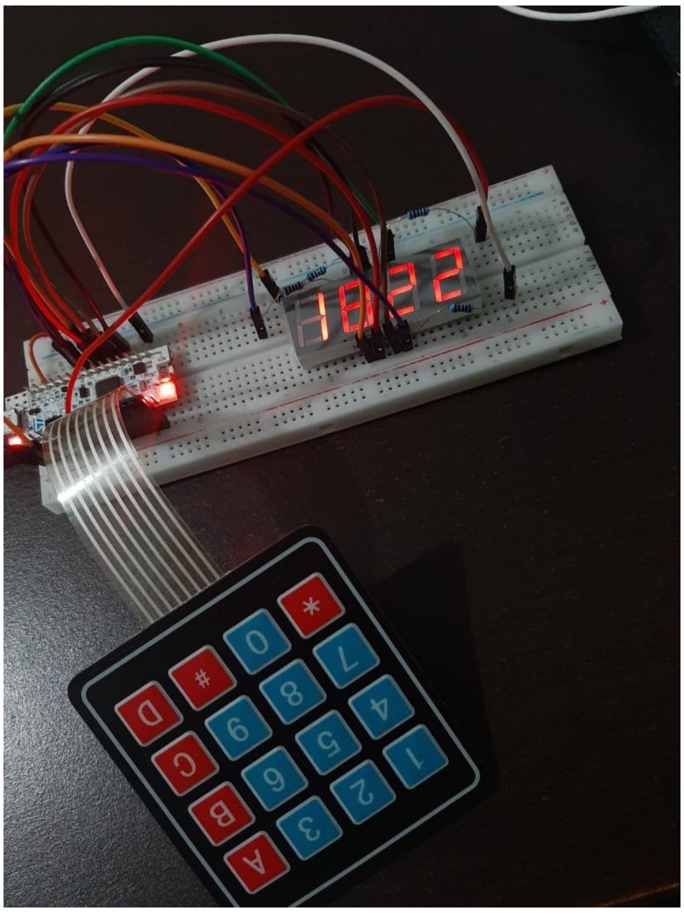


Figure 6. Project setup photo

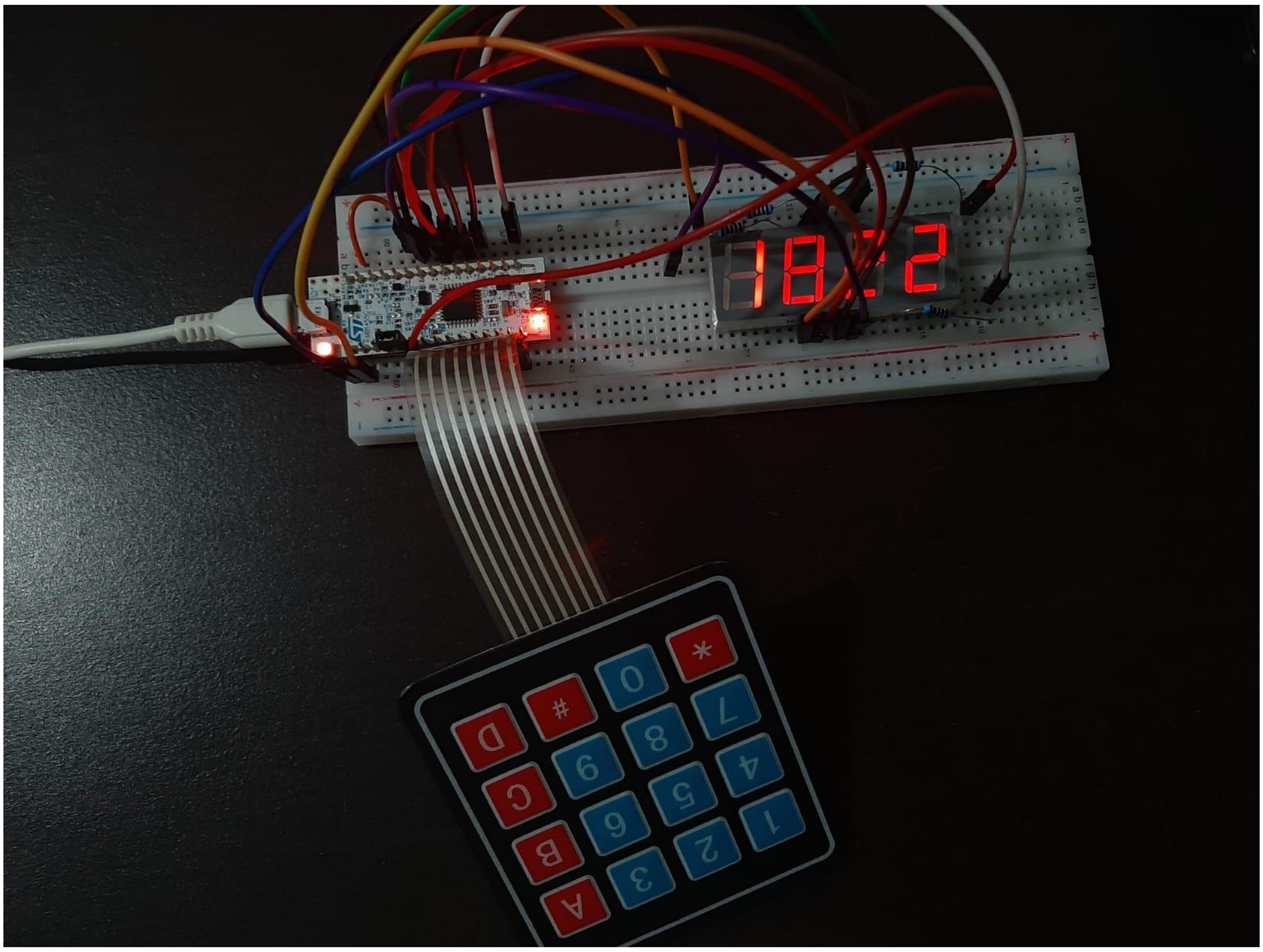


Figure 7. Project setup photo

The circuit used in the project setup is as in the photos seen Figure 5,6,7. Connections between board and SSD, board and keypad has been made by using Figure 1,2,3,4 are as follows.

A → A0,	B → A1,	C → D7,	D → A3,	E → A4,	F → A5,
G → A6,	Digit1 → D1,	Digit2 → D0,	Digit3 → D3,	Digit4 → D13,	
R1 → D12,	R2 → D11,	R3 → D10,	R4 → D9,	C1 → D8,	C2 → D7,
C3 → D6,	C4 → D5 ,	DP (decimal point) → A7			

4. Tasks

4.1. Task I – Display the pressed button

As described in theoretical research, the keypad works according to the logic of short-circuiting the pressed button and checking the rows for 5V or 0V. For this, all relevant connections were established as mentioned in Project Setup. Digits were powered and when the desired digit was activated, it was aimed to see the number in that digit.

In order for the microprocessor to fit the operating logic and not to engage the microprocessor with many functions and commands, the buttons of the keypad were determined as interrupt external functions on the code. In this way, when the appropriate code was written, the desired button was pressed and that number could be seen on the SSD.

The functions used and their duties in short are as follows:

```
void keypad_read(void);
```

This function performs operations such as activation of ports A and B, setting input and output modes for the rows and columns of the keypad, activation of related registers for external interrupt, determining priority levels and activation of them in the NVIC.

```
void EXTI0_1_IRQHandler(void);
void EXTI2_3_IRQHandler(void);
void EXTI4_15_IRQHandler(void);
```

These three functions are external interrupt functions. Pressing the button performs a 5V or 0V check of the relevant pin and the symbol of the letter, number or sign with the button pressed according to this decision is displayed on the SSD.

```
void moders_for_digits();
```

This function activates the relevant pins for digits.

```
void clearSSD(void);
void setSSD(int x);
```

These two functions perform the process of cleaning the SSD screen or adjusting the desired number.

```
void clear_Rows_Keypad(void);
void set_Rows_Keypad(void);
```

These two functions make the row pins on the keypad pull to 5V or 0V.

```
void odr_for_digit1();
void odr_for_digit2();
void odr_for_digit3();
void odr_for_digit4();
```

Each of these four functions provides the process of activation and powering for each of the digits.

The tasks specified in all these functions are carried out in the appropriate order. When the button is pressed from keypad, the external interrupt function with the pressed button is displayed, and after the necessary checks are made here, the relevant icon is displayed on the SSD screen with the `setSSD` function.

For example, when pressing 5 via keypad, the image on the SSD can be seen in Figure 8.

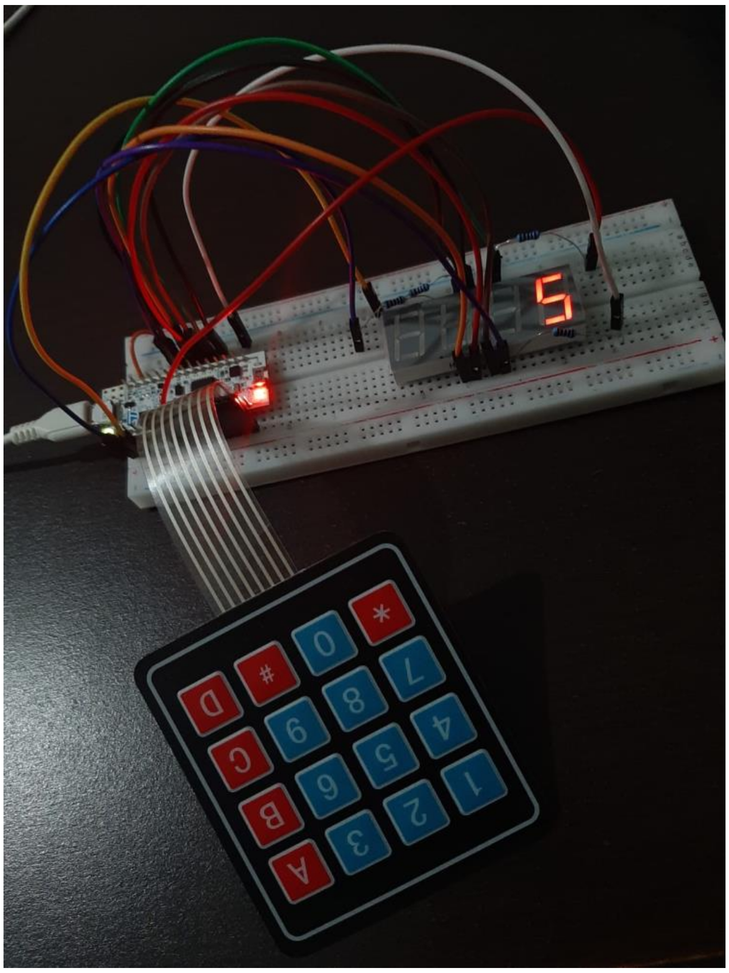


Figure 8. Task 1 example photo

Interpretation of Results:

Targeted success was achieved in this task. There was no very challenging situation in terms of setting up an algorithm. The important thing here was that the keypad working logic was thoroughly grasped and the 5V or 0V control accordingly was done for each row pin.

Since the buttons had to be examined in different external interrupt functions within themselves, the relevant controls had to be carried out in three different functions. The need to check one by one for each state in each function led to confusion and some errors at some points.

For solutions to errors and confusions, problems were resolved when the if/else if/else control blocks button was pressed and used in accordance with the 5V or 0V status.

4.2. Task II – Displaying school number

In this task, the progress made in the previous task has been continued. Another function has been added next to the functions used. This function will show the first two and last two digits of the school number on the SSD in the initial state when the circuit is run. The prototype of the function is as follows:

```
void school_id();
```

The process of showing the first two and last two digits of the school number will be carried out by assigning short-term delay and calling the desired numbers with the help of `setSSD` function in a loop. A loop that will turn up to a certain number will ensure that the numbers remain on the screen continuously. In order for this operation to be performed, the logic of activate-deactivate digits will be used.

One digit is activated and the other digits are deactivated at the same time, and this process is repeated for each digit. Related numbers for related digits are called with the `setSSD` function. Again, for the continuity of this process, small delay assignment procedures should be made on the lines where the activate-deactivate operations is located. In this way, the desired number will be visible continuously on the SSD.

When we look at Figure 9, it is seen that this task is built on the circuit.

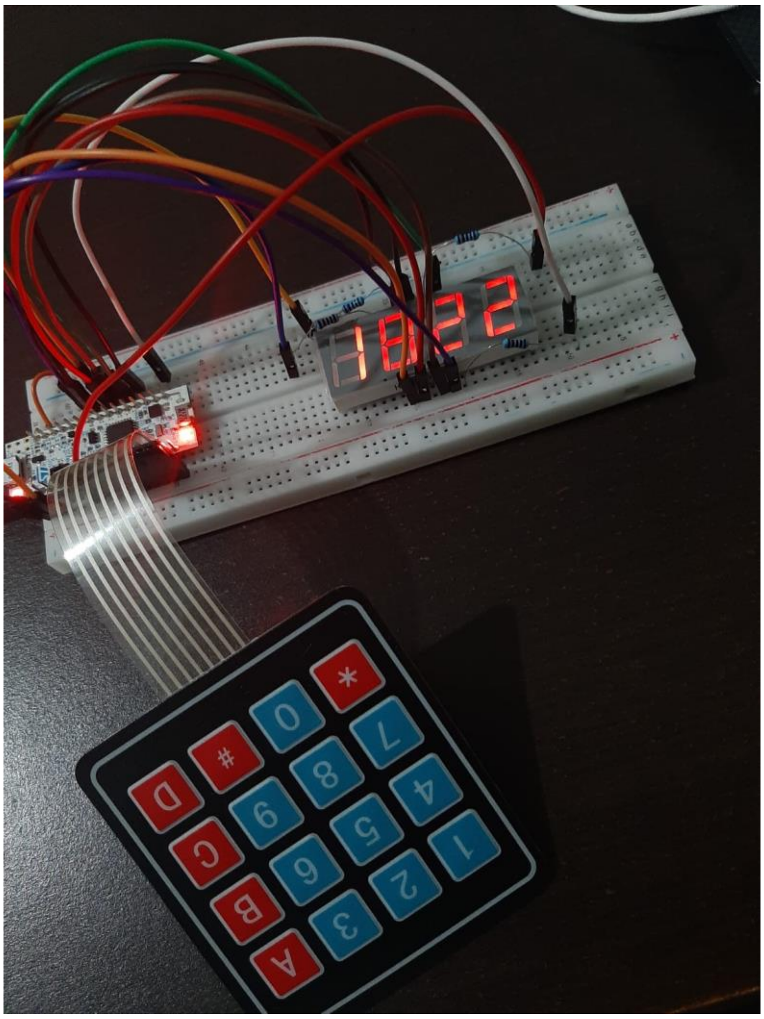


Figure 9. Task 2 example photo

Interpretation of Results:

The first two and last two digits of the school number were successfully displayed on the screen thanks to the specified function. Activate-deactivate digits logic was the basic building block in doing this operation.

The problem encountered at this point was the place where this function was called in main. After the pins, which are normally used for digits, the `school_id` function was called and `keypad_read` function was called afterwards. Meanwhile, it was seen that the code was not working and time was spent here for a long time. Afterwards, `keypad_read` was noticed that this caused problems on the basis of the `school_id` function, as many other clocks and registers such as pin activation and mod adjustment were performed within the function. When this function (`keypad_read`) called before the `school_id` function, in other words, when the `school_id` function called after the keypad_read function, the problem was solved.

4.3. Task III – Mathematical operations

In this task, an algorithm on how to perform mathematical operations is established and based on this algorithm for all operations. This algorithm can be examined as follows:

As soon as the button is pressed via keypad, the relevant external interrupt function is taken. The number printed with a `number` variable created in memory, that is, the number is thrown into the number variable. If addition, subtraction, multiplication, and division are made, the value in the number variable is thrown to the variable named `number_catch` the `number` variable by going to the relevant case block. Thus, in the event of pressing the next button, the second number can be kept in the variable `number`, while the first number `number_catch` in the variable named number. If a scientific or trigonometric calculation is performed, it will be enough to maintain the number in a single variable because one number will already be processed.

Each of the numbers is subject to a **and** operation with 0x7FFFFFFF to avoid the problem of obtaining incorrect results. Each process has its own variable. The result function for this variable to get a value of 1 is taken, where the display is performed on the SSD.

The function of each key on keypad is as follows:

A → addition, logarithm base 10, sin

B → subtraction, natural logarithm, cos

C → multiplication, square root, tan

D → division, square, cot

# → scientific mode

## → trigonometric mode

→ π

After scientific and trigonometric modes, whichever letter is pressed will go to the case block containing the relevant process, as shown in the order above.

A separate result function was used for the results of the operations performed in trigonometric and scientific modes. This is due to the comma of the results. The decimal point part of the SSD was also used for comma numbers. A different algorithm has been developed to see the comma number on the screen. For example, if the number is 2,197 (ln9), this number is first multiplied by 1000 and sent to its result function in this way. Here, the digits are separated. In the result function, the relevant if/else if block is entered according to the range of values received by the number, where it is determined that the point will be placed on the SSD. For the sampled number, the point will be placed in the part where the first digit is located. If the number was a negative number such as -0.95 (sin5), then the control block would be entered for negative numbers and the result of the operations here would be placed in the part where the point second digit is located.

Since the value of the number Π is specific (3.141), a separate imaging function is written for this number and the point is placed directly in the part where the first digit is located.

Here are brief descriptions of the functions added and used for this task:

```
void print_final(int t);
```

This function is the result function for addition, subtraction, multiplication and division. The function parameter number it receives is used to display the change on the SSD.

```
void print_final_dot(int t);
```

This function is used for the results of scientific and trigonometric calculations.

```
void print_final_pi(int t);
```

This function is used for the results of Π number on the SSD.

```
void SSD_activate(void);
```

Added to activate SSD when needed.

```
void digit1_SSD(int x);
```

```
void digit2_SSD(int x);
void digit3_SSD(int x);
```

```
void digit4_SSD(int x);
```

These functions activate their digits and turn off the others and show the value they receive on the screen with the values they receive and the `setSSD` functions they contain.

```
void moder_activate_dot();
```

```
void moder_deactivate_dot();
```

These functions are used to activate and deactivate the decimal point connected to the PA7 pin as output.

The following are illustrations of ln9 (2,197) and sin5 results on the SSD.

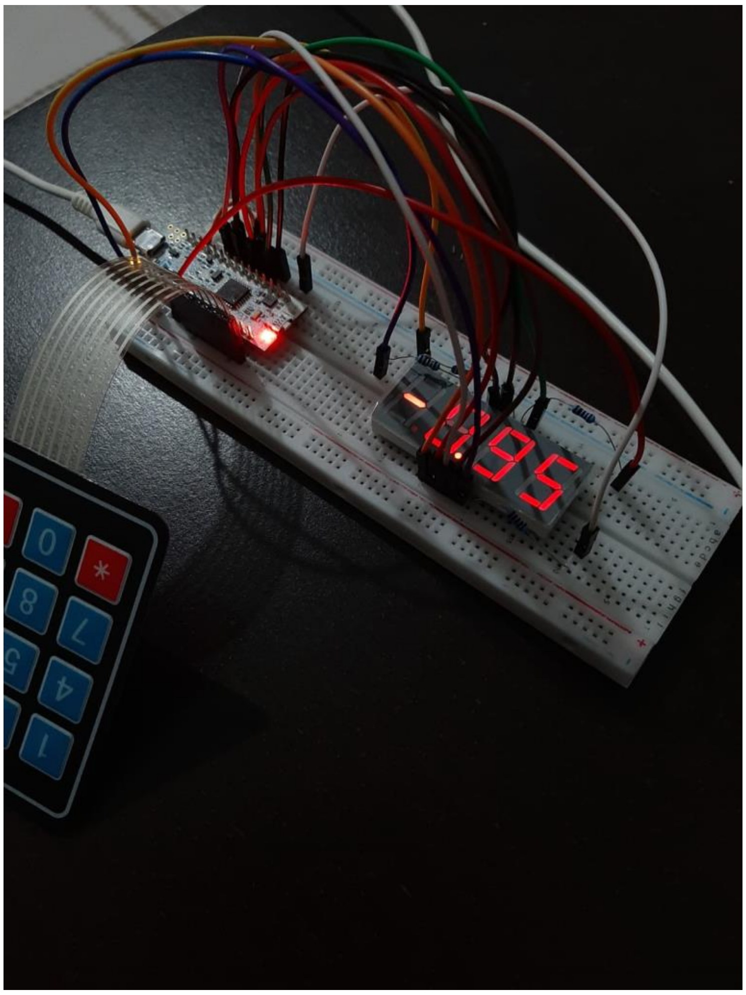


Figure 10. Task 3 example photo (sin5)

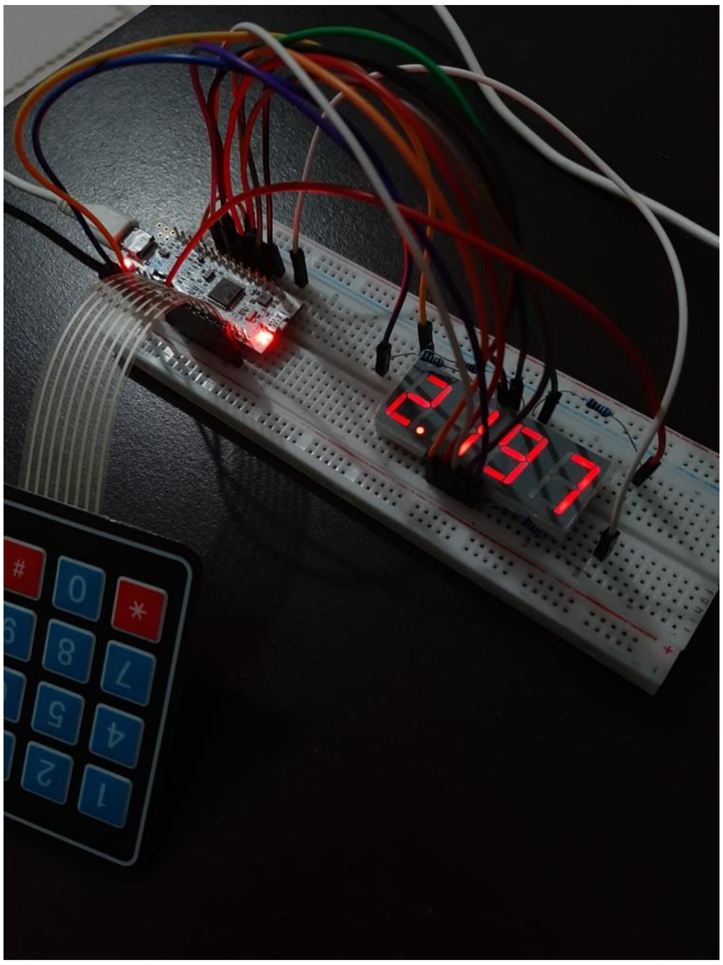


Figure 11. Task 3 example photo (ln9)

Interpretation of Results:

In this task, the requests were done successfully to a certain extent. When single digit numbers were entered on the keypad for all operations, it was seen that the correct result was reached. Some of the requirements that have not been achieved or are considered incomplete are as follows:

This was not achieved when processing was requested even if a number entry with more than one digit was received on keypad. The number was obtained on the desired digit as output. However, unfortunately, this situation was not achieved when it came to input. An argument to solve this situation was made with a `control` variable. Although the double digit number could be seen when the status of the numbers was tracked with breakpoint, the desired results could not be seen on the SSD screen. That's why only single-digit numbers remained available.

There are some difficulties in being able to process without intervention via computer. These problems do not include addition, subtraction, multiplication, and division operations. However, in order for scientific and trigonometric modes to be processed, it was noticed that breakpoint assignments should be made to the relevant locations when the code is run. For example, for logarithm, a breakpoint must be placed in the case block (case 15) with the # icon and the case block (case 11) with the B icon. In this way, the desired result was obtained when the code started and stopped. To solve this problem, button bouncing solving algorithms have been tried in the relevant regions but have not been successful.

4.4. Task IV – Invalid operations and overflow

In this task, it is emphasized that there are situations that are mathematically incorrect and are not physically possible.

Invalid operations:

For example, the 0 division of any number is not mathematically correct. If such an entry comes from the user side, it is shown on the SSD screen that this situation is incorrect.

Overflow:

It consists of four digits on the 4-digit SSD name. Therefore, it will not be possible to show a value greater than 9999 or less than -999.

The functions used to display these states on the screen are as follows:

```
void invalid_inputs();

void overflow_inputs();
```

In the event that invalid operation is an input, the SSD screen will be as in the photo below:

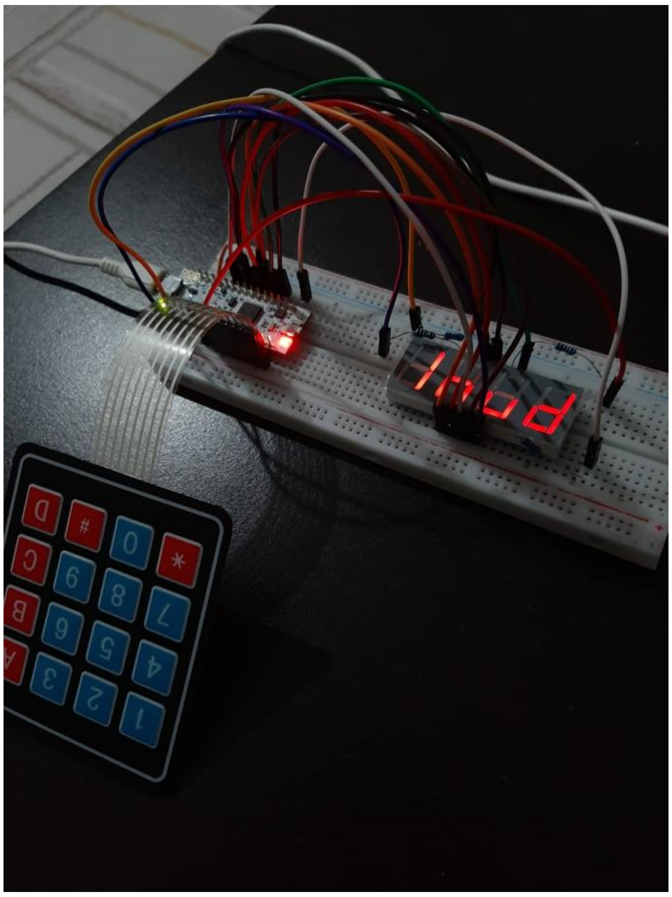
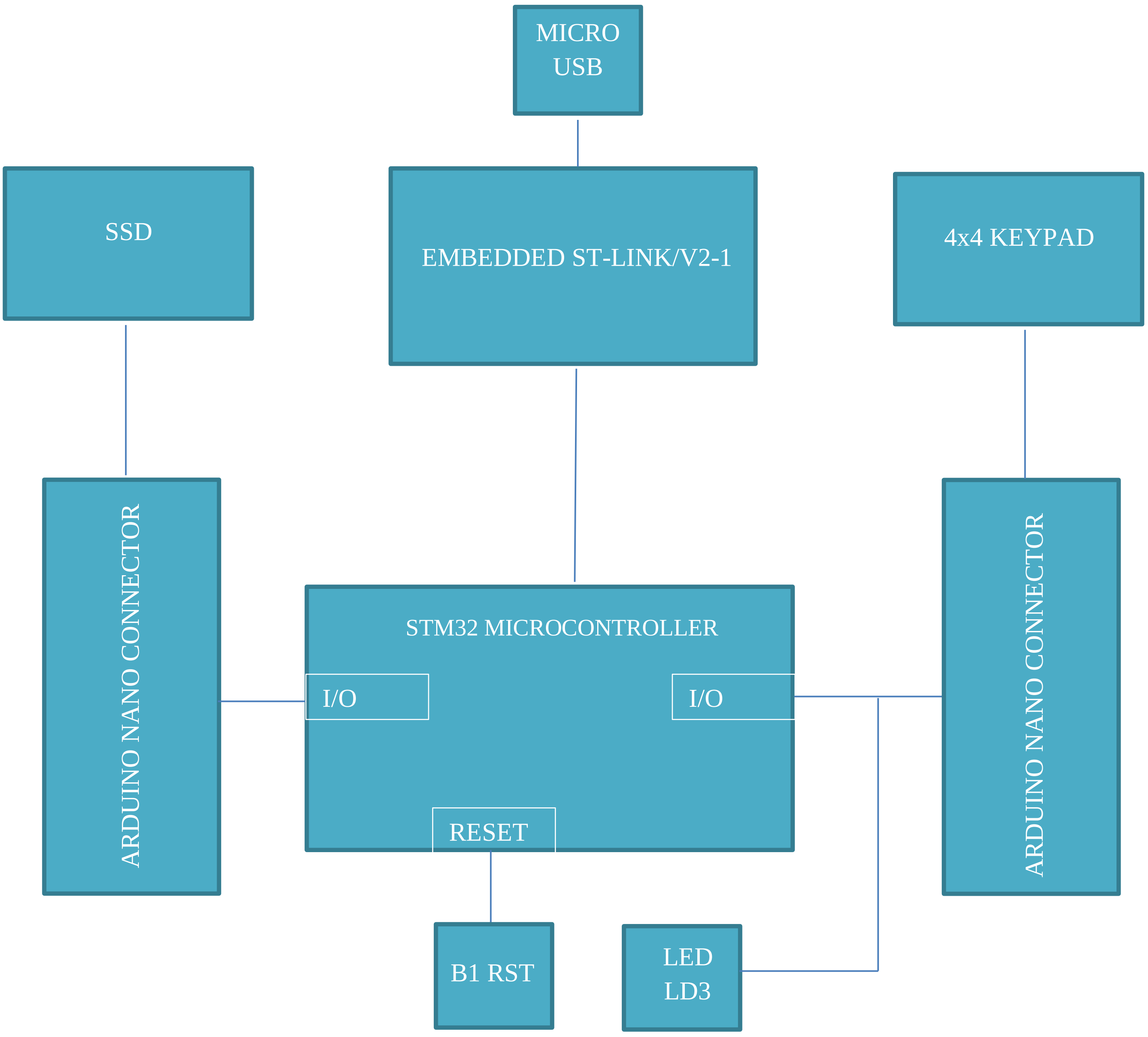


Figure 12. Task 4 example photo (invalid op.)

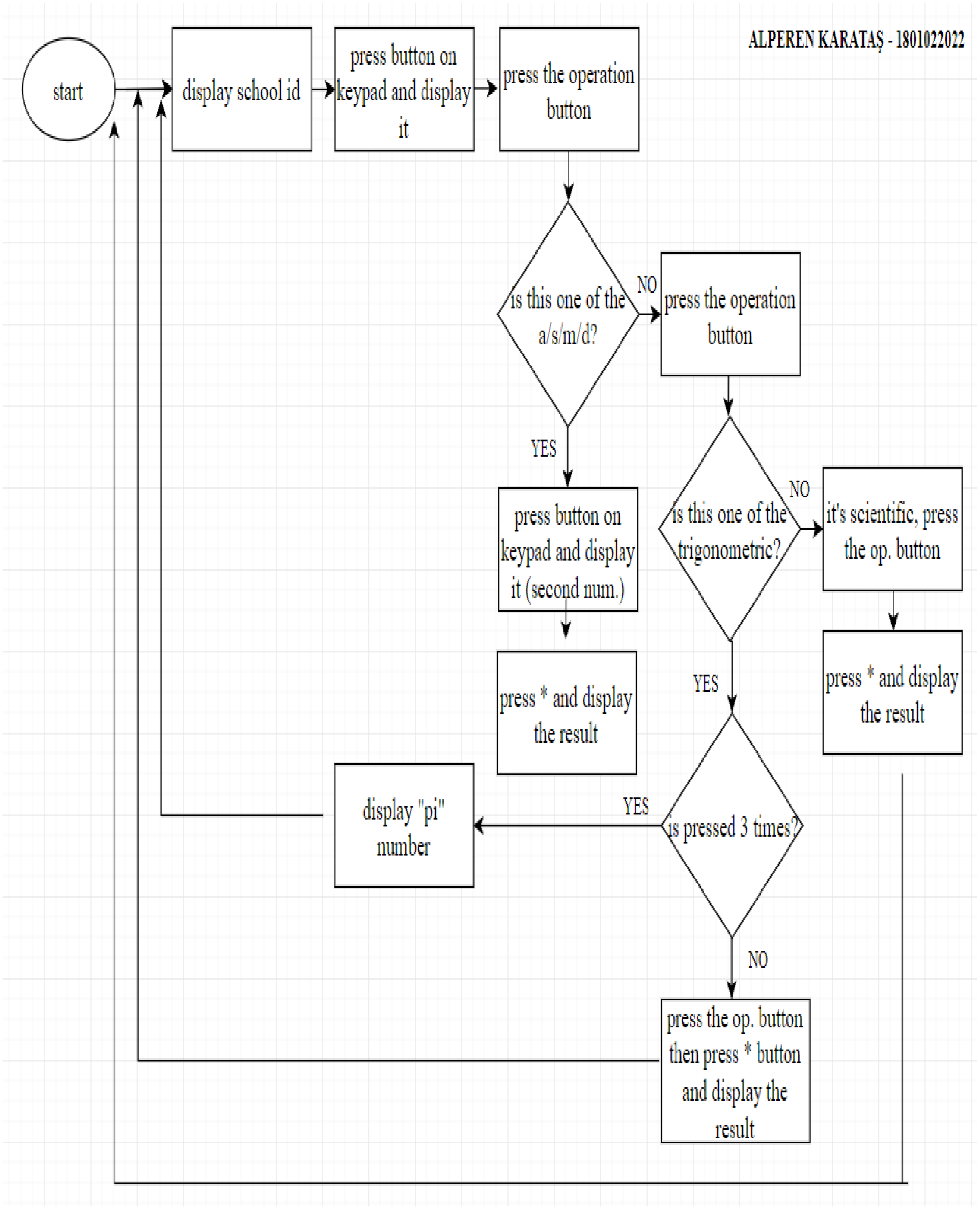
Interpretation of Results:

No compelling situations or setbacks have been encountered in this task. The desired success will be achieved when using the appropriate if/else if/else blocks to check invalid operations and overflow statuses. Functions were used to print these articles on the screen as indicated.

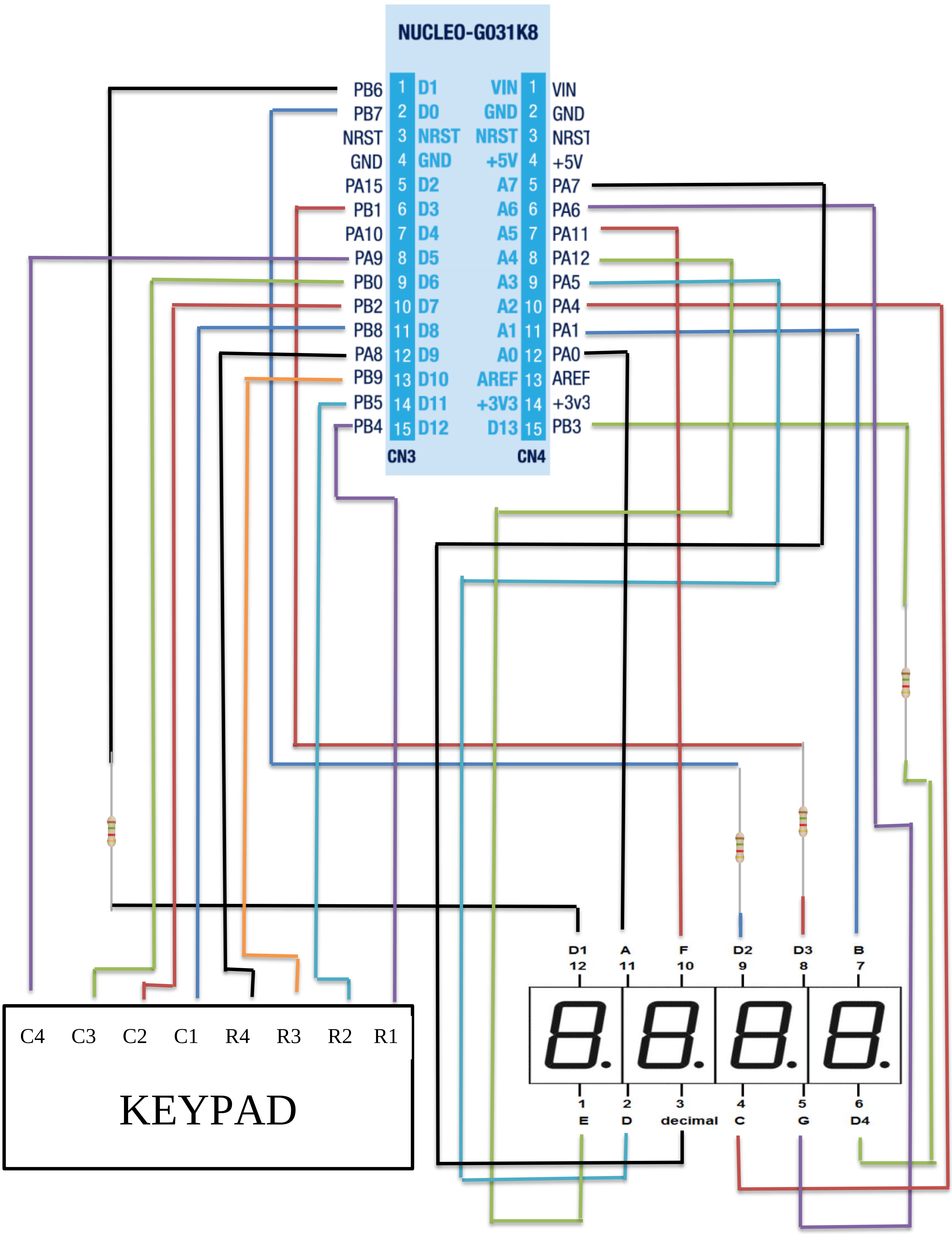
Block Diagram:



Flowchart:


ALPEREN KARATAŞ - 1801022022
start
display school id
press button on keypad and display it
press the operation button
is this one of the a/s/m/d?
NO
press the operation button
YES
press button on keypad and display it (second num.)
press * and display the result
is this one of the trigonometric?
NO
it's scientific, press the op. button
press * and display the result
YES
is pressed 3 times?
YES
display "pi" number
NO
press the op. button then press * button and display the result

Schematic Diagram:



Regarding circuit and code, block diagram, flowchart and schematic diagram are as seen in the above ways.

5. Results and General Comments

In order to progress comfortably in the project and to understand and implement what was done, reviews and applications were carried out on 4 different task. Each task depends on the previous task in another when examined in order. Taking it one step at a time is the main path. When one completed successfully, the other was switched.

This project has provided great benefits in solving the problems encountered and using the ability to think. A great improvement has been achieved in the ability to establish algorithms. Although the requirements seem to be long and difficult, when they are analyzed piece by piece, it is understood that the problems can be solved and the requirements can be achieved. Practical solutions have been brought to the problems, and it is fully understood how the calculator can be designed as hardware.

As detailed in the **"Interpretation of Results "** sections, there have been some problems in some parts, some of which may be solved, but some unfortunately have not. However, since these problems did not disrupt the main idea of the project and the goal of achieving what was desired, the project was completed **successfully** in general.

6. Tasks List

Tasks	**Completion Status**
I. Display the pressed button	✓
II. Displaying school number	✓
III. Mathematical operations	✓
IV. Invalid operations and overflow	✓

7. Parts List

Row Number	Product name	Piece	Unit Price	Total Price
1	STM32 Nucleo G031K8	1 piece	127,16 TL	104,52 TL (with discount)
2	Single breadboard	1 piece	7,44 TL	7,44 TL
3	14mm 4 Digit Common Anode Seven Segment Display	1 piece	6,26 TL	6,26 TL
4	4x4 Membran Keypad	1 piece	4,70 TL	4,70 TL
5	1K Resistor	4 pieces	0,03 TL	0,12 TL
6	Male-Male Jumper Cable	11-12 pieces	0,13 TL	1,56 TL

8. References

[1] RM0444 Reference manual STM32G0x1 advanced Arm®-based 32-bit MCUs

[2] https://github.com/fcayci

[3] https://www.direnc.net/ 14mm 4-digit Common Anode Seven Segment Display Datasheet

[4] https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet

[5] https://www.hepteknoloji.net/ (Resistor image)

5. Youtube Links

[1] https://youtu.be/1xXK4KBbMII

Appendix A. Code Listing

```
/* project2.c
*
* Prepared by:
* Alperen Karatas - 1801022022
*
* Notes:
* ELEC334 2020 Fall - Project 2
* Fully Operational Scientific Calculator
*
*/

#include "stm32g0xx.h"
#include "math.h"
#include "bsp.h"

int main(void) {

	moders_for_digits();
	SSD_activate();
	keypad_read();
	school_id();
	clearSSD();

    while(1) {

    }

    return 0;
}
```

```
/* project2.h
*
* Prepared by:
* Alperen Karataş - 1801022022
*
*/

#ifndef BSP_H_
#define BSP_H_

void delay_ms(volatile uint32_t s);
void keypad_read(void);
void delay(volatile uint32_t s);
void clearSSD(void);
void setSSD(int x);
void print_final(int t);
void print_final_dot(int t);
void print_final_pi(int t);
void invalid_inputs();
void overflow_inputs();
void clear_Rows_Keypad(void);
void set_Rows_Keypad(void);
void moders_for_digits();
void school_id();
void odr_for_digit1();
void odr_for_digit2();
void odr_for_digit3();
void odr_for_digit4();
void SSD_activate(void);
void digit1_SSD(int x);
void digit2_SSD(int x);
void digit3_SSD(int x);
void digit4_SSD(int x);
void moder_activate_dot();
void moder_deactivate_dot();
void initTIM3(void);

#endif
```

```c
/* bsp.c
*
* Prepared by:
* Alperen Karataş - 1801022022
*
*/
 _Bool button_state = 1;

volatile int number=0;
volatile int number_catch=0;
int control=0;

volatile int is_in_add=0;
volatile int is_in_ext=0;
volatile int is_in_mul=0;
volatile int is_in_div=0;

volatile int is_in_sci=0;
volatile int is_in_sci_log10=0;
volatile int is_in_sci_ln=0;
volatile int is_in_sci_root=0;
volatile int is_in_sci_square=0;

volatile int is_in_tri=-1;      // attention, it's initial value is -1 because
of double pressing for trigonometric mode
volatile int is_in_tri_sin=0;
volatile int is_in_tri_cos=0;
volatile int is_in_tri_tan=0;
volatile int is_in_tri_cot=0;
volatile float is_in_tri_pi=0;

volatile int final=0;
volatile float final_sci_tri=0.0;

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

static int count =0;

void SysTick_Handler(void){
	count++;
}

void delay_ms(volatile uint32_t s) {
    count = 0;
    while(count < s);
}
void keypad_read(void){
			// Enable GPIOA clock
			    RCC->IOPENR |= (1U << 0);

			// Enable GPIOB clock
			    RCC->IOPENR |= (1U << 1);

			// Setup pa8, pb9, pb5, pb4 as output rows
			    GPIOA->MODER &= ~(3U << 2*8);
			    GPIOA->MODER |= (1U << 2*8);
```

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

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

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

             //Setup pa9, pb0, pb2, pb8 as input columns
                 GPIOA->MODER &= ~(3U << 2*9);
                 GPIOA->PUPDR |= (2U << 2*9);

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

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

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

             // Setup interrupts for pa9, pb0,pb2,pb8
                 EXTI->EXTICR[2] |= (0U << 8*1); //pa9
                 EXTI->EXTICR[0] |= (1U << 0);   //pb0
                 EXTI->EXTICR[0] |= (1U << 8*2); //pb2
                 EXTI->EXTICR[2] |= (1U << 0);   //pb8

             //rising edge selected for pa9, pb0,pb2,pb8
                 EXTI->RTSR1 |= (1U << 9);
                 EXTI->RTSR1 |= (1U << 0);
                 EXTI->RTSR1 |= (1U << 2);
                 EXTI->RTSR1 |= (1U << 8);

             //mask register selected for pa9, pb0,pb2,pb8
                 EXTI->IMR1 |= (1U << 9);
                 EXTI->IMR1 |= (1U << 0);
                 EXTI->IMR1 |= (1U << 2);
                 EXTI->IMR1 |= (1U << 8);

             // enable NVIC and set interrupt priority
                 NVIC_SetPriority(EXTI0_1_IRQn, 0);
                 NVIC_EnableIRQ(EXTI0_1_IRQn);

                 NVIC_SetPriority(EXTI2_3_IRQn, 0);
                 NVIC_EnableIRQ(EXTI2_3_IRQn);

                 NVIC_SetPriority(EXTI4_15_IRQn, 0);
                 NVIC_EnableIRQ(EXTI4_15_IRQn);

             /* Setup PA0 as output */
                   GPIOA->MODER &= ~(3U << 2 * 0);
                   GPIOA->MODER |= (1U << 2 * 0);
                   /* Setup PA1 as output */
                   GPIOA->MODER &= ~(3U << 2 * 1);
                   GPIOA->MODER |= (1U << 2 * 1);
                   /* Setup PA4 as output */
                   GPIOA->MODER &= ~(3U << 2 * 4);
                   GPIOA->MODER |= (1U << 2 * 4);
```

```
/* Setup PA5 as output */
                   GPIOA->MODER &= ~(3U << 2 * 5);
                   GPIOA->MODER |= (1U << 2 * 5);
                   /* Setup PA6 as output */
                   GPIOA->MODER &= ~(3U << 2 * 6);
                   GPIOA->MODER |= (1U << 2 * 6);
                   /* Setup PA11 as output */
                   GPIOA->MODER &= ~(3U << 2 * 11);
                   GPIOA->MODER |= (1U << 2 * 11);
                   /* Setup PA12 as output */
                   GPIOA->MODER &= ~(3U << 2 * 12);
                   GPIOA->MODER |= (1U << 2 * 12);



                   GPIOA -> ODR |= (1U << 8);  //PA0 A
                   GPIOB -> ODR |= (1U << 9);  //PA1 B
                   GPIOB -> ODR |= (1U << 5);  //PA4 C
                   GPIOB -> ODR |= (1U << 4);  //PA5 D

                   clearSSD();
}

void EXTI0_1_IRQHandler(void){ //interrupt from pb0
      clear_Rows_Keypad();

      GPIOA->ODR ^= (1U << 8);
      if ((GPIOB -> IDR >>0)& 1){
            setSSD(15); //#
      }
      GPIOA->ODR ^= (1U << 8); //pa8


      GPIOB->ODR ^= (1U << 9); //pb9
      if ((GPIOB -> IDR >>0)& 1){
                   setSSD(9);
                   number=number*pow(10,control);
                   number+=9;
                   control=1;
            }
      GPIOB->ODR ^= (1U << 9); //pb9

GPIOB->ODR ^= (1U << 5); //pb5
            if ((GPIOB -> IDR >>0)& 1){
                         setSSD(6);
                         number=number*pow(10,control);
                         number+=6;
                         control=1;
                   }
      GPIOB->ODR ^= (1U << 5); //pb5

GPIOB->ODR ^= (1U << 4); //pb4
            if ((GPIOB -> IDR >>0)& 1){
                         setSSD(3);
                         number=number*pow(10,control);
                         number+=3;
                         control=1;
                   }
      GPIOB->ODR ^= (1U << 4); //pb4
```

```
EXTI->RPR1 |= (1U << 0); //clear interrupt flag
        set_Rows_Keypad();

}

void EXTI2_3_IRQHandler(void){ //interrupt from PB2
        clear_Rows_Keypad();
        GPIOA->ODR ^= (1U << 8);
        if((GPIOB->IDR >>2) & 1){
                setSSD(0);
                number=number*pow(10,control);
                number+=0;
                control=1;
        }
        GPIOA->ODR ^= (1U << 8); //pa8


        GPIOB->ODR ^= (1U << 9); //pb9
        if((GPIOB->IDR >>2) & 1){
                setSSD(8);
                number=number*pow(10,control);
                number+=8;
                control=1;
        }
        GPIOB->ODR ^= (1U << 9); //pb9


        GPIOB->ODR ^= (1U << 5); //pb5
        if((GPIOB->IDR >>2) & 1){
                setSSD(5);
                number=number*pow(10,control);
                number+=5;
                control=1;
        }
        GPIOB->ODR ^= (1U << 5); //pb5

        GPIOB->ODR ^= (1U << 4); //pb4
        if((GPIOB->IDR >>2) &1){
                setSSD(2);
                number=number*pow(10,control);
                number+=2;
                control=1;
        }
        GPIOB->ODR ^= (1U << 4); //pb4

        EXTI->RPR1 |= (1U << 2); //clear interrupt flag
        set_Rows_Keypad();
}

void EXTI4_15_IRQHandler(void){ //pb8 and pa9
        if((GPIOB->IDR >>8) &1){

                clear_Rows_Keypad();
                GPIOA->ODR ^= (1U << 8);
                if((GPIOB->IDR >>8) &1){
                        setSSD(14);

                }
                GPIOA->ODR ^= (1U << 8); //pa8
```

```
		GPIOB->ODR ^= (1U << 9); //pb9
		if((GPIOB->IDR >>8) &1){
			setSSD(7);
			number=number*pow(10,control);
			number+=7;
			control=1;
		}
		GPIOB->ODR ^= (1U << 9); //pb9


		GPIOB->ODR ^= (1U << 5); //pb5
		if((GPIOB->IDR >>8) &1){
			setSSD(4);
			number=number*pow(10,control);
			number+=4;
			control=1;
		}
		GPIOB->ODR ^= (1U << 5); //pb5

		GPIOB->ODR ^= (1U << 4); //pb4
		if((GPIOB->IDR >>8) &1){
			setSSD(1);
			number=number*pow(10,control);
			number+=1;
			control=1;
		}
		GPIOB->ODR ^= (1U << 4); //pb4
		EXTI->RPR1 |= (1U << 8); //clear interrupt flag
		set_Rows_Keypad();
	}
	if((GPIOA->IDR >>9) &1){
		clear_Rows_Keypad();
		GPIOA->ODR ^= (1U << 8);
		if((GPIOA->IDR >>9) &1){
			setSSD(13);
		}
		GPIOA->ODR ^= (1U << 8); //pa8

		GPIOB->ODR ^= (1U << 9); //pb9
		if((GPIOA->IDR >>9) &1){
			setSSD(12);
		}
		GPIOB->ODR ^= (1U << 9); //pb9

		GPIOB->ODR ^= (1U << 5); //pb5
		if((GPIOA->IDR >>9) &1){
			setSSD(11);
		}
		GPIOB->ODR ^= (1U << 5); //pb5


		GPIOB->ODR ^= (1U << 4); //pb4
		if((GPIOA->IDR >>9) &1){
			setSSD(10);
		}
		GPIOB->ODR ^= (1U << 4); //pb4
		EXTI->RPR1 |= (1U << 9); //clear interrupt flag
		set_Rows_Keypad();}

		}
```

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

void clearSSD(void){

	GPIOA -> ODR |= (1U << 0);  //PA0 A
	GPIOA -> ODR |= (1U << 1);  //PA1 B
	GPIOA -> ODR |= (1U << 4);  //PA4 C
	GPIOA -> ODR |= (1U << 5);  //PA5 D
	GPIOA -> ODR |= (1U << 12); //PA12 E
	GPIOA -> ODR |= (1U << 11); //PA11 F
	GPIOA -> ODR |= (1U << 6);  //PA6 G
}


void setSSD(int x){
	clearSSD();
	switch(x){
		case 0:
			GPIOA -> ODR &= ~(1U << 0);  //PA0 A
			GPIOA -> ODR &= ~(1U << 1);  //PA1 B
			GPIOA -> ODR &= ~(1U << 4);  //PA4 C
			GPIOA -> ODR &= ~(1U << 5);  //PA5 D
			GPIOA -> ODR &= ~(1U << 12); //PA12 E
			GPIOA -> ODR &= ~(1U << 11); //PA11 F
			break;

		case 1:
			GPIOA -> ODR &= ~(1U << 1);  //PA1 B
			GPIOA -> ODR &= ~(1U << 4);  //PA4 C
			break;

		case 2:
			GPIOA -> ODR &= ~(1U << 0);  //PA0 A
			GPIOA -> ODR &= ~(1U << 1);  //PA1 B
			GPIOA -> ODR &= ~(1U << 5);  //PA5 D
			GPIOA -> ODR &= ~(1U << 12); //PA12 E
			GPIOA -> ODR &= ~(1U << 6);  //PA6 G
			break;

		case 3:
			GPIOA -> ODR &= ~(1U << 0);  //PA0 A
			GPIOA -> ODR &= ~(1U << 1);  //PA1 B
			GPIOA -> ODR &= ~(1U << 4);  //PA4 C
			GPIOA -> ODR &= ~(1U << 5);  //PA5 D
			GPIOA -> ODR &= ~(1U << 6);  //PA6 G
			break;
		case 4:
			GPIOA -> ODR &= ~(1U << 1);  //PA1 B
			GPIOA -> ODR &= ~(1U << 4);  //PA4 C
			GPIOA -> ODR &= ~(1U << 11); //PA11 F
			GPIOA -> ODR &= ~(1U << 6);  //PA6 G
			break;
```

```
case 6:
                    GPIOA -> ODR &= ~(1U << 0);  //PA0 A
                    GPIOA -> ODR &= ~(1U << 4);  //PA4 C
                    GPIOA -> ODR &= ~(1U << 5);  //PA5 D
                    GPIOA -> ODR &= ~(1U << 12); //PA12 E
                    GPIOA -> ODR &= ~(1U << 11); //PA11 F
                    GPIOA -> ODR &= ~(1U << 6);  //PA6 G
                    break;

             case 7:
                    GPIOA -> ODR &= ~(1U << 0);  //PA0 A
                    GPIOA -> ODR &= ~(1U << 1);  //PA1 B
                    GPIOA -> ODR &= ~(1U << 4);  //PA4 C
                    break;

             case 8:
                    GPIOA -> ODR &= ~(1U << 0);  //PA0 A
                    GPIOA -> ODR &= ~(1U << 1);  //PA1 B
                    GPIOA -> ODR &= ~(1U << 4);  //PA4 C
                    GPIOA -> ODR &= ~(1U << 5);  //PA5 D
                    GPIOA -> ODR &= ~(1U << 12); //PA12 E
                    GPIOA -> ODR &= ~(1U << 11); //PA11 F
                    GPIOA -> ODR &= ~(1U << 6);  //PA6 G
                    break;

             case 9:
                    GPIOA -> ODR &= ~(1U << 0);  //PA0 A
                    GPIOA -> ODR &= ~(1U << 1);  //PA1 B
                    GPIOA -> ODR &= ~(1U << 4);  //PA4 C
                    GPIOA -> ODR &= ~(1U << 5);  //PA5 D
                    GPIOA -> ODR &= ~(1U << 11); //PA11 F
                    GPIOA -> ODR &= ~(1U << 6);  //PA6 G
            break;

             /* A -> addition - logarithm base 10 - sin */
             case 10:
                    if(number!=0 && is_in_sci==0 && is_in_tri==-1){
                           number &= 0x7FFFFFFF;
                           number_catch = number;
                           is_in_add++;
                           number = 0;
                    }
                    if(is_in_tri==1){
                           number &= 0x7FFFFFFF;
                           is_in_tri_sin++;
                    }
                    else if(is_in_sci==1){
                           number &= 0x7FFFFFFF;
                           is_in_sci_log10++;
                    }
                    break;
             /* B -> extraction - natural logarithm (ln) - cos */
             case 11:
                    if(number!=0 && is_in_sci==0 && is_in_tri==-1){
                           number &= 0x7FFFFFFF;
                           number_catch = number;
                           is_in_ext++;
                           number = 0;
}
                    if(is_in_tri==1){
                          number &= 0x7FFFFFFF;
                          is_in_tri_cos++;
                    }
                    else if(is_in_sci==1){
                          number &= 0x7FFFFFFF;
                          is_in_sci_ln++;
                    }
                    break;

              /* C -> multiplication - square root - tan */
              case 12:
                    if(number!=0 && is_in_sci==0 && is_in_tri==-1){
                          number &= 0x7FFFFFFF;
                          number_catch = number;
                          is_in_mul++;
                          number = 0;
                    }
                    if(is_in_tri==1){
                          number &= 0x7FFFFFFF;
                          is_in_tri_tan++;
                    }
                    else if(is_in_sci==1){
                          number &= 0x7FFFFFFF;
                          is_in_sci_root++;
                    }
                    break;

              /* D -> division - square - cot */
              case 13:
                    if(number!=0 && is_in_sci==0 && is_in_tri==-1){
                          number &= 0x7FFFFFFF;
                          number_catch = number;
                          is_in_div++;
                          number = 0;

                    }
                    if(is_in_tri==1){
                          number &= 0x7FFFFFFF;
                          is_in_tri_cot++;
                    }

                    else if(is_in_sci==1){
                          number &= 0x7FFFFFFF;
                          is_in_sci_square++;
                    }
                    break;

                    /* * -> final results */

              case 14:
                    if(is_in_add==1){
                          if (number!=0)
                                number &= 0x7FFFFFFF;
                          final = number_catch+number;
                          print_final(final);
                    }
```

```
if(is_in_ext==1){
                                if (number!=0)
                                        number &= 0x7FFFFFFF;
                                final = number_catch-number;
                                print_final(final);
                        }
                        if(is_in_mul==1){
                                if (number!=0)
                                        number &= 0x7FFFFFFF;
                                final = number_catch*number;
                                if(final < -999 || final > 9999){      // overflow
                                        overflow_inputs();
                                }
                                else{
                                        print_final(final);
                                }

                        }
                        if(is_in_div==1){
                                if (number!=0)
                                        number &= 0x7FFFFFFF;
                                final = number_catch/number;
                                if(number==0){
                                        invalid_inputs();
                                }
                                else{
                                        print_final(final);
                                }
                        }
                        if(is_in_sci_log10==1){
                                final_sci_tri=log10(number);
                                final=final_sci_tri*1000;
                                print_final_dot(final);
                        }
                        if(is_in_sci_ln==1){
                                final_sci_tri=log(number);
                                final=final_sci_tri*1000;
                                print_final_dot(final);
                        }
                        if(is_in_sci_root==1){
                                final_sci_tri=sqrt(number);
                                if(number<0){                   // invalid
                                        invalid_inputs();
                                }
                                else{
                                        final=final_sci_tri*1000;
                                        print_final(final);
                                }
                        }

                if(is_in_sci_square==1){
                                final=pow(number,2);
                                print_final(final);
                        }
```

```c
if(is_in_tri_sin==1){
					final_sci_tri=sin(number);
					final=final_sci_tri*1000;
					print_final_dot(final);
				}
				if(is_in_tri_cos==1){
					final_sci_tri=cos(number);
					final=final_sci_tri*1000;
					print_final_dot(final);
				}
				if(is_in_tri_tan==1){
					final_sci_tri=tan(number);
					final=final_sci_tri*1000;
					print_final_dot(final);
				}
				if(is_in_tri_cot==1){
					final_sci_tri=1/tan(number);
					final=final_sci_tri*1000;
					print_final_dot(final);
				}
				break;

			/* # -> scientific - if pressed three times it will show the
number of pi */
			case 15:
				if(is_in_tri==1){
					final=3141;               // pi=3.141
					print_final_pi(final);
				}
				is_in_sci++;
				is_in_tri++;
				break;

			/* O */
			case 16:
				GPIOA -> ODR &= ~(1U << 4);  //PA2 C
				GPIOA -> ODR &= ~(1U << 5);  //PA3 D
				GPIOA -> ODR &= ~(1U << 12); //PA4 E
				GPIOA -> ODR &= ~(1U << 6);  //PA6 G
				break;

			/* u */
			case 17:
				GPIOA -> ODR &= ~(1U << 4);  //PA2 C
				GPIOA -> ODR &= ~(1U << 5);  //PA3 D
				GPIOA -> ODR &= ~(1U << 12); //PA4 E
				break;

			/* f */
			case 18:
				GPIOA -> ODR &= ~(1U << 0);  //PA0 A
				GPIOA -> ODR &= ~(1U << 12); //PA4 E
				GPIOA -> ODR &= ~(1U << 11); //PA5 F
				GPIOA -> ODR &= ~(1U << 6);  //PA6 G
				break;
```

```
/* L */
            case 19:
                  GPIOA -> ODR &= ~(1U << 5);  //PA3 D
                  GPIOA -> ODR &= ~(1U << 12); //PA4 E
                  GPIOA -> ODR &= ~(1U << 11); //PA5 F
                  break;

            /* I */
            case 20:
                  GPIOA -> ODR &= ~(1U << 1);  //PA1 B
                  GPIOA -> ODR &= ~(1U << 4);  //PA4 C
                  break;

            /* n */
            case 21:
                  GPIOA -> ODR &= ~(1U << 6);  //PA6 G
                  GPIOA -> ODR &= ~(1U << 12); //PA12 E
                  GPIOA -> ODR &= ~(1U << 4);  //PA4 C
                  break;

            /* u */
            case 22:
                  GPIOA -> ODR &= ~(1U << 12); //PA12 E
                  GPIOA -> ODR &= ~(1U << 4);  //PA4 C
                  GPIOA -> ODR &= ~(1U << 5);  //PA3 D
                  break;

            /* d */
            case 23:
                  GPIOA -> ODR &= ~(1U << 6);  //PA6 G
                  GPIOA -> ODR &= ~(1U << 12); //PA12 E
                  GPIOA -> ODR &= ~(1U << 4);  //PA4 C
                  GPIOA -> ODR &= ~(1U << 5);  //PA3 D
                  GPIOA -> ODR &= ~(1U << 1);  //PA1 B
                  break;

/* minus(-) sign */
            case 24:
                  odr_for_digit1();
                  moder_deactivate_dot();
                  GPIOA -> ODR &= ~(1U << 6);  //PA6 G
                  delay(500);
                  break;

            /* decimal point (dot) */
            case 25:
                  GPIOA -> ODR &= ~(1U << 7);  //PA7 DP
                  delay(500);
                  break;
      }
}
```

```c
/* addition, extraction, multiplication and division print function for SSD */
void print_final(int t){
	if(t<0){
		t=t*(-1);
		int x=t/1000;
		int y=(t/100)%10;
		int z=(t/10)%10;
		int k=t%10;
		for(int i=0; i<6500; ++i){
			setSSD(24);
			digit2_SSD(x);
			digit3_SSD(y);
			digit4_SSD(z);
		}
		school_id();        // back to idle state
	}
	 else{
		 int x=t/1000;
		 int y=(t/100)%10;
		 int z=(t/10)%10;
		 int k=t%10;
		 for(int i=0; i<6500; ++i){
			digit1_SSD(x);
			digit2_SSD(y);
			digit3_SSD(z);
			digit4_SSD(k);
		}
		 school_id();        // back to idle state
	}
}

/* scintific/trigonometric (float numbers) print function for SSD */
void print_final_dot(int t){
	if(t<0){
		t=t*(-1);
		int x=t/1000;
		int y=(t/100)%10;
		int z=(t/10)%10;
		int k=t%10;
		if(t>0 && t<10000){
			for(int i=0; i<6500; ++i){
				odr_for_digit2();
				moder_activate_dot();
				setSSD(25);
				setSSD(24);
				digit2_SSD(x);
				digit3_SSD(y);
				digit4_SSD(z);
			}
			school_id();        // back to idle state
		}
else if(t>=10000 && t<=100000){
                   for(int i=0; i<6500; ++i){
                         odr_for_digit3();
                         moder_activate_dot();
                         setSSD(25);
                         setSSD(24);
                         digit2_SSD(x);
                         digit3_SSD(y);
                         digit4_SSD(z);
                   }
             }
             school_id();        // back to idle state
       }
        else{
              int x=t/1000;
              int y=(t/100)%10;
              int z=(t/10)%10;
              int k=t%10;
              if(t>0 && t<10000){
                    for(int i=0; i<6500; ++i){
                          odr_for_digit1();
                          moder_activate_dot();
                          setSSD(25);
                          digit1_SSD(x);
                          digit2_SSD(y);
                          digit3_SSD(z);
                          digit4_SSD(k);
                    }
                    school_id();        // back to idle state
              }
              else if(t>=10000 && t<=100000){
                    for(int i=0; i<6500; ++i){
                          odr_for_digit2();
                          moder_activate_dot();
                          setSSD(25);
                          digit1_SSD(x);
                          digit2_SSD(y);
                          digit3_SSD(z);
                          digit4_SSD(k);
                    }
                    school_id();        // back to idle state
              }

else if(t>=10000 && t<=100000){
                    for(int i=0; i<6500; ++i){
                          odr_for_digit3();
                          moder_activate_dot();
                          setSSD(25);
                          digit1_SSD(x);
                          digit2_SSD(y);
                          digit3_SSD(z);
                          digit4_SSD(k);
                    }
              }
              school_id();            // back to idle state
       }
}
```

```c
/* pi number print function for SSD */
void print_final_pi(int t){
              int x=t/1000;
              int y=(t/100)%10;
              int z=(t/10)%10;
              int k=t%10;
              for(int i=0; i<6500; ++i){
                   odr_for_digit1();
                   moder_activate_dot();
                   digit1_SSD(x);
                   digit2_SSD(y);
                   digit3_SSD(z);
                   digit4_SSD(k);
              }
              school_id();         // back to idle state
}

void invalid_inputs(){
int ctr_inv=0;
while(ctr_inv<10000){
      /* I */
      odr_for_digit1();
      setSSD(20);
      delay(500);
      /* n */
      odr_for_digit2();
      setSSD(21);
      delay(500);
      /* u */
      odr_for_digit3();
      setSSD(22);
      delay(500);
      /* d */
      odr_for_digit4();
      setSSD(23);
      delay(500);
      ctr_inv++;
}
      clearSSD();
}

void overflow_inputs(){
int ctr_ovr=0;
while(ctr_ovr<10000){
      /* O */
      odr_for_digit1();
      setSSD(16);
      delay(500);
      /* u */
      odr_for_digit2();
      setSSD(17);
      delay(500);
      /* F */
      odr_for_digit3();
      setSSD(18);
      delay(500);
      /* L */
      odr_for_digit4();
      setSSD(19);
delay(500);
        ctr_ovr++;
}
        clearSSD();

}

void clear_Rows_Keypad(void){
        GPIOA -> ODR &= ~(1U << 8);  //PA8
        GPIOB -> ODR &= ~(1U << 9);  //PB9
        GPIOB -> ODR &= ~(1U << 5);  //PB5
        GPIOB -> ODR &= ~(1U << 4);  //PB4
}

void set_Rows_Keypad(void){
        GPIOA -> ODR |= (1U << 8);  //PA8
        GPIOB -> ODR |= (1U << 9);  //PB9
        GPIOB -> ODR |= (1U << 5);  //PB5
        GPIOB -> ODR |= (1U << 4);  //PB4
}

void moders_for_digits(){
        RCC->IOPENR |= (1U << 1);
        GPIOB->MODER &= ~(3U << 2*1);
        GPIOB->MODER |= (1U << 2*1);
        GPIOB->MODER &= ~(3U << 2*3);
        GPIOB->MODER |= (1U << 2*3);
        GPIOB->MODER &= ~(3U << 2*6);
        GPIOB->MODER |= (1U << 2*6);
        GPIOB->MODER &= ~(3U << 2*7);
        GPIOB->MODER |= (1U << 2*7);
}
void school_id(){
int ctr_id=0;
while(ctr_id<5000){
        odr_for_digit1();
        setSSD(1);
        delay(500);
        odr_for_digit2();
        setSSD(8);
        delay(500);
        odr_for_digit3();
        setSSD(2);
        delay(500);
        odr_for_digit4();
        setSSD(2);
        delay(500);
        ctr_id++;
}
clearSSD();
}
```

```c
void odr_for_digit1(){
      GPIOB ->ODR |= (1U << 6);   // D1  digit to PB6
      GPIOB ->ODR &= ~(1U << 7);  // D2  digit to PB7
      GPIOB ->ODR &= ~(1U << 1);  // D3  digit to PB1
      GPIOB ->ODR &= ~(1U << 3);  // D4  digit to PB3
}
void odr_for_digit2(){
      GPIOB ->ODR &= ~(1U << 6);
      GPIOB ->ODR |= (1U << 7);
      GPIOB ->ODR &= ~(1U << 1);
      GPIOB ->ODR &= ~(1U << 3);
}
void odr_for_digit3(){
      GPIOB ->ODR &= ~(1U << 6);
      GPIOB ->ODR &= ~(1U << 7);
      GPIOB ->ODR |= (1U << 1);
      GPIOB ->ODR &= ~(1U << 3);
}
void odr_for_digit4(){
      GPIOB ->ODR &= ~(1U << 6);
      GPIOB ->ODR &= ~(1U << 7);
      GPIOB ->ODR &= ~(1U << 1);
      GPIOB ->ODR |= (1U << 3);
}

void SSD_activate(void){
      GPIOB ->ODR |= (1U << 6);
      GPIOB ->ODR |= (1U << 7);
      GPIOB ->ODR |= (1U << 1);
      GPIOB ->ODR |= (1U << 3);
}

void digit1_SSD(int x){
      GPIOB ->ODR |=   (1U << 6);
      GPIOB ->ODR &=  ~(1U << 7);
      GPIOB ->ODR &=  ~(1U << 1);
      GPIOB ->ODR &=  ~(1U << 3);
      setSSD(x);
      delay(500);
      clearSSD();
}
void digit2_SSD(int x){
      moder_deactivate_dot();
      GPIOB ->ODR &=  ~(1U << 6);
      GPIOB ->ODR |=   (1U << 7);
      GPIOB ->ODR &=  ~(1U << 1);
      GPIOB ->ODR &=  ~(1U << 3);
      setSSD(x);
      delay(500);
      clearSSD();
}
```

```
void digit3_SSD(int x){
	moder_deactivate_dot();
	GPIOB ->ODR &=  ~(1U << 6);
	GPIOB ->ODR &=  ~(1U << 7);
	GPIOB ->ODR |=   (1U << 1);
	GPIOB ->ODR &=  ~(1U << 3);
	setSSD(x);
	delay(500);
	clearSSD();
}

void digit4_SSD(int x){
	moder_deactivate_dot();
	GPIOB ->ODR &=  ~(1U << 6);
	GPIOB ->ODR &=  ~(1U << 7);
	GPIOB ->ODR &=  ~(1U << 1);
	GPIOB ->ODR |=   (1U << 3);
	setSSD(x);
	delay(500);
	clearSSD();
}

void moder_activate_dot(){
	/* Setup PA7 as output */
	GPIOA->MODER &= ~(3U << 2 * 7);
	GPIOA->MODER |= (1U << 2 * 7);
}

void moder_deactivate_dot(){
	/* Setup PA7 as output */
	GPIOA->MODER &= ~(3U << 2 * 7);
}
```