GEBZE TECNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC 334 - Project #2

A Fully Operational Scientific Calculator

Project #2 REPORT

Preparer:
1) 171024027 – Mehmet Akif GÜMÜŞ

1. INRODUCTION:

Main objective of this project/midterm is to create a fully operational scientific calculator in C. This calculator will have a keypad connected to enter the numbers and execute basic scientific and trigonometric functions. A 4-digit seven segment display should be used to display these numbers.

2. Technical requirements:

- Written in C. No HAL or equivalent libraries.
- A keypad and a seven-segment display should be attached
- On power up SSD should show your ID (first 2 and last 2 digits).
    - As soon as a number is pressed, everything should be cleared and only your number should be displayed
    - When keys are entered, the SSD should shift the numbers to the left, while not displaying anything for empty digits.
- If the digits are already full, new number key presses should be ignored.
- ABCDEF keys should be used as:
    - A is for addition
    - B is for subtraction
    - C is for multiplication
    - D is for division
    - E key is scientific mode, and will expect another keypress.
        - EA is for log
        - EB is for In
        - EC is for sqrt
        - ED is for x^2
        - EE is for trigonometric mode, and will expect another keypress.
            - EEA is for sin
            - EEB is for cos
            - EEC is for tan
            - EED is for cot
            - EEE is for pi
    - F key is for enter/equal
- Scientific and trigonometric modes will require floating point number system.
    - Floating point numbers should be displayed with the appropriate dot. For example if you want to show 1.2345152 - SSD should display 1.234 and if you want to display 4213.123 it should display 4213.
- Negative numbers should have a negative sign. i.e-124 on the SSD.
- If the numbers overflows 9999 or -999, it should display overflow (i.e. OuFL)
- If the operation is invalid (i.e. 3/0 or sqrt(-2)) it should display invalid (i.e. Invd)
- If no keys are pressed for 10 seconds, the SSD should turn off. - go back to IDLE state.

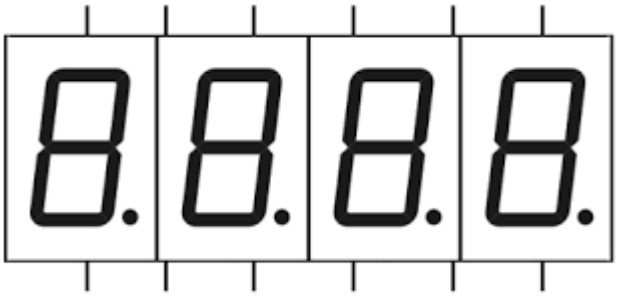
If directly a function is invoked, the current value should be used. For example, if the last answer is 4 and - 4 is pressed, it should do 4 - 4 operation and display 0. If in the beginning, the number should be assumed 0.

THEORETICAL RESEARCH

- ### Seven-segment display

A **seven-segment display** is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.



Şekil 1. example of 4-digit 7 segment display

- ### KeyPAD

The 4*4 matrix keypad usually is used as input in a project. It has 16 keys in total, which means the same input values.

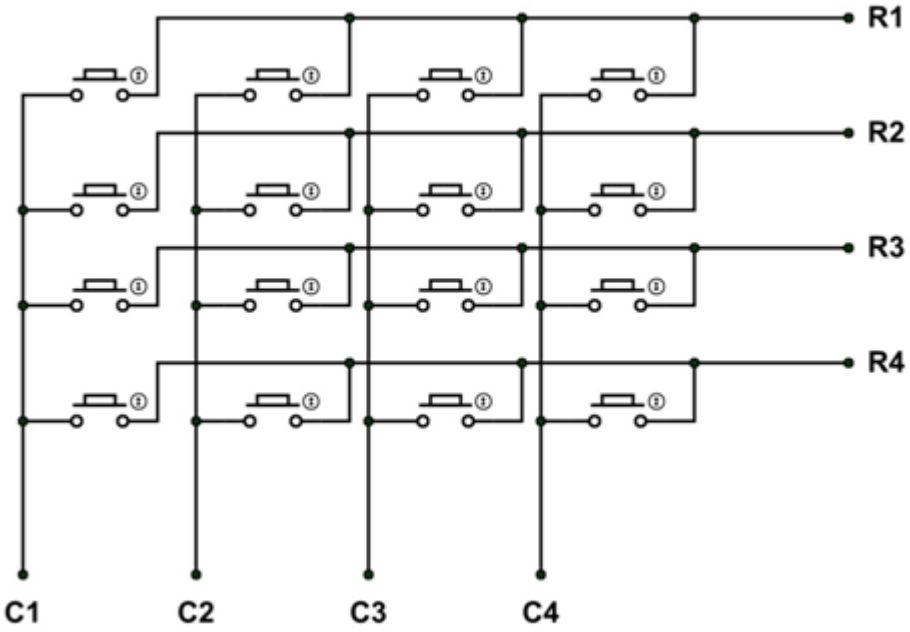
The SunFouner 4*4 Matrix Keypad Module is a matrix non- encoded keypad consisting of 16 keys in parallel. The keys of each row and column are connected through the pins outside – pin Y1-Y4 as labeled beside control the rows, when X1-X4, the columns.

How it works

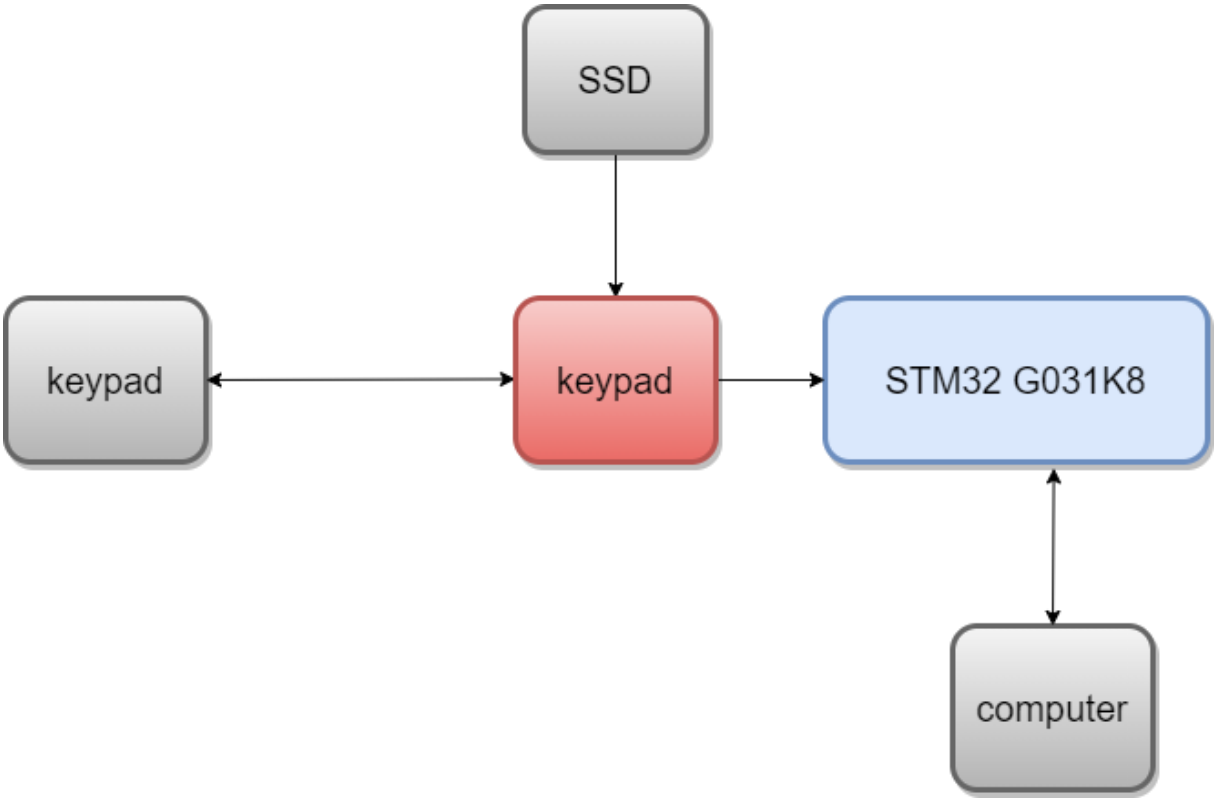
First test whether any key is pressed down. Connect power to rows, so they are High level. Then set all the rows Y1-Y4 as Low and then detect the status of the columns. Any column of Low indicates there is key pressing and that the key is among the 4 keys of the column. If all columns are High, it means no key is pressed down.

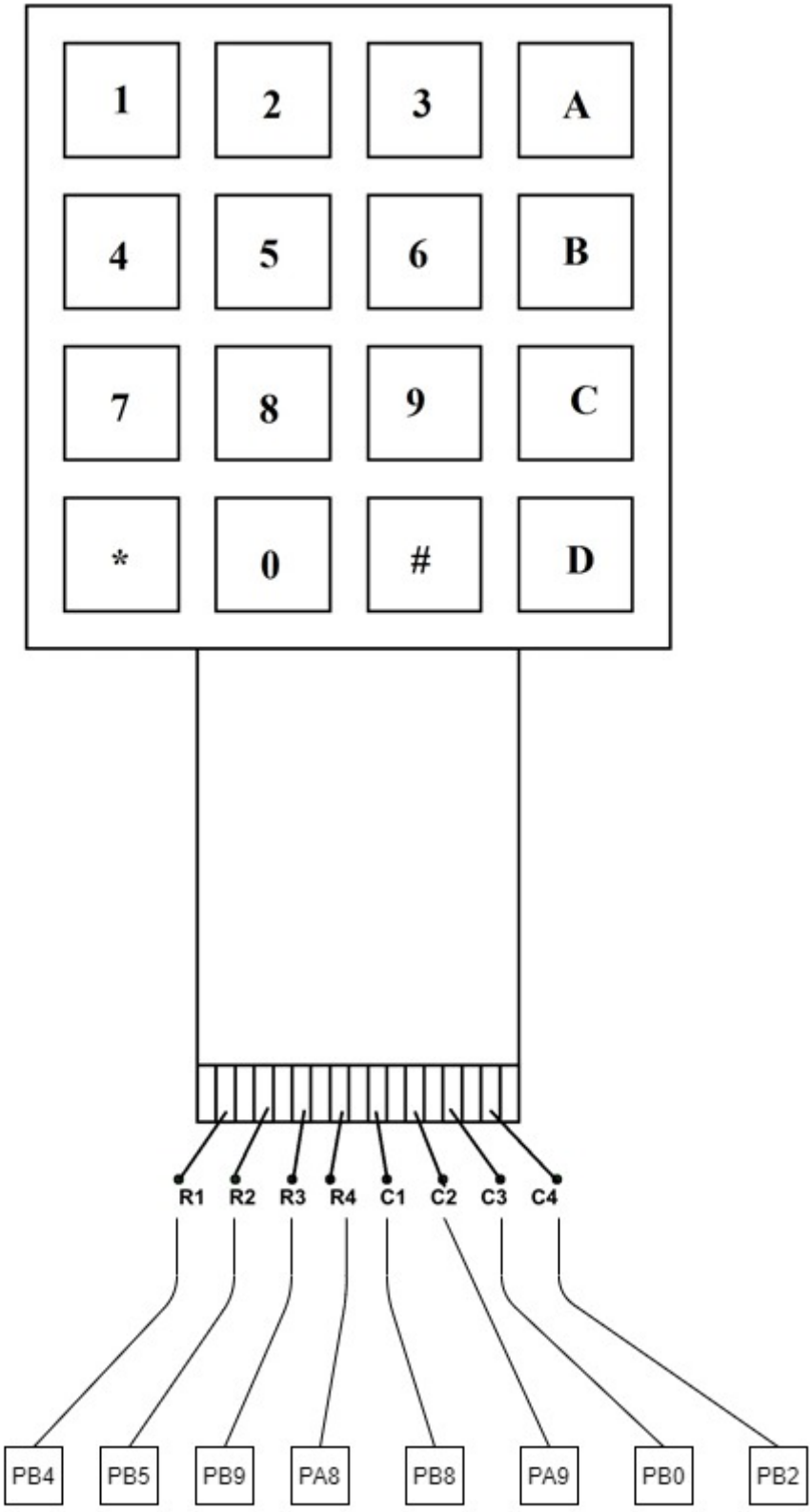
Next, locate the key. Since the column in which the pressed key lies is identified, knowing the line would finalize the testing. Thus, set the rows as Low in turns until any is unveiled accordingly – other rows will still be High. Now the row can be identified. Detect the status of each column in turns. The column tested Low is the one intersecting with the line – their cross point is just the key pressed.

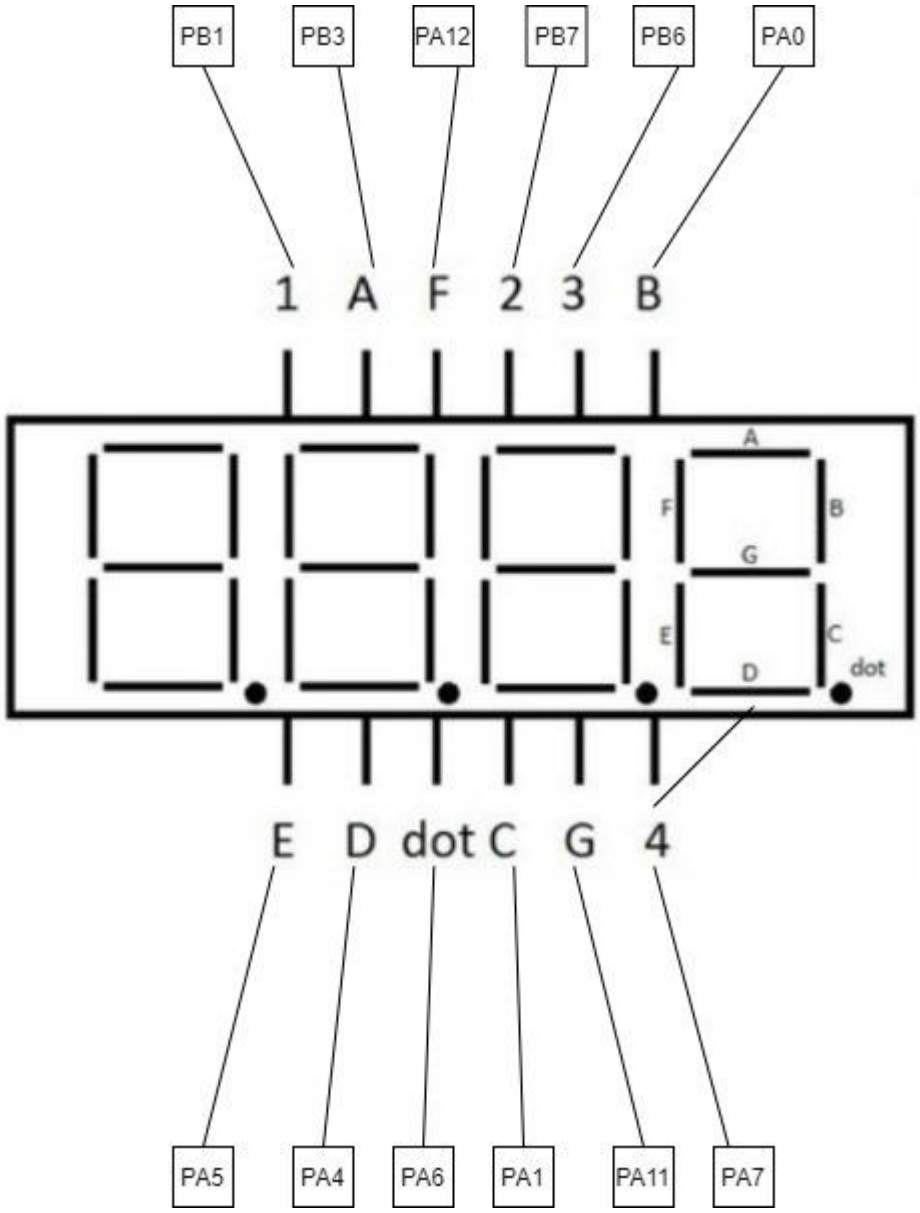
The schmatic diagram:



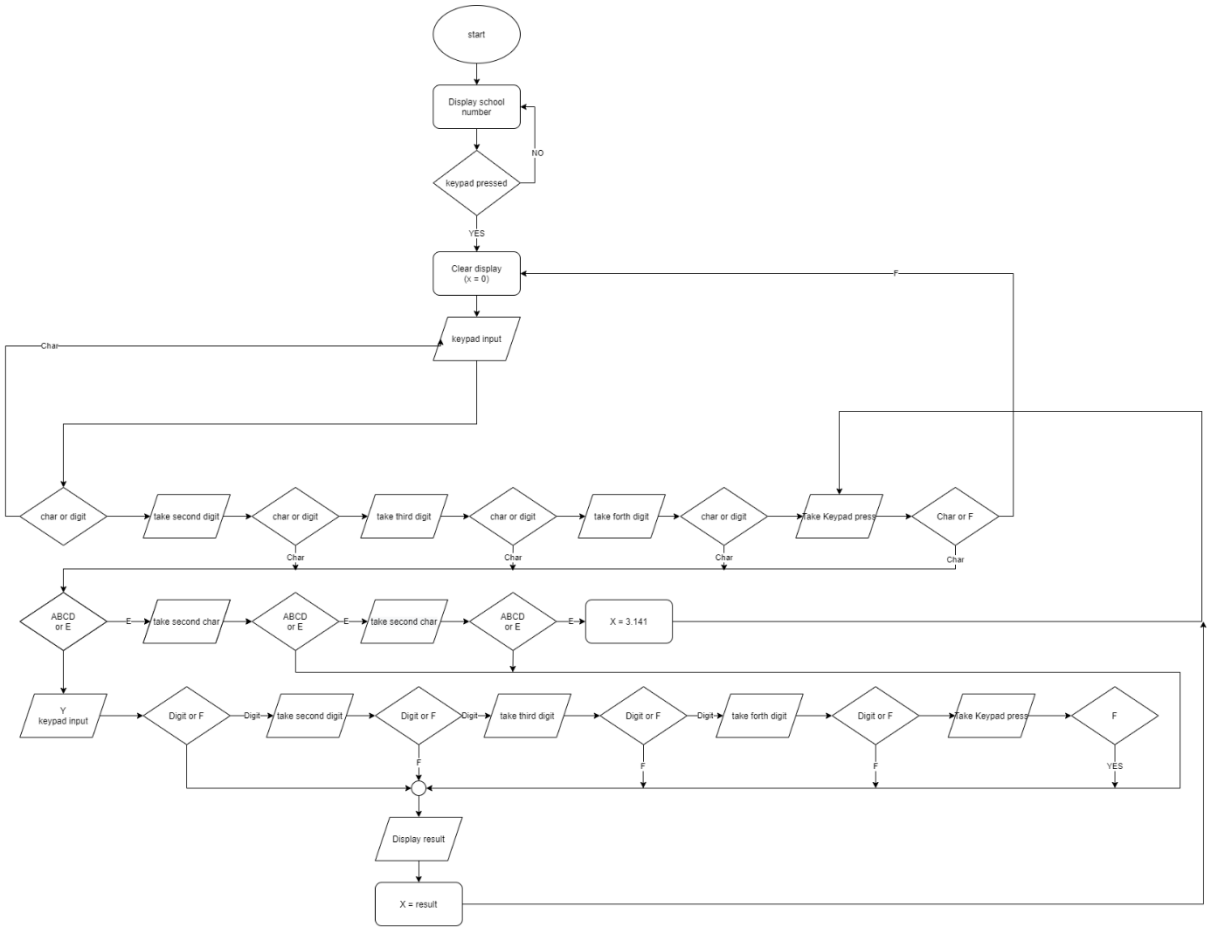
3. BLOCK DIAGRAM




1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D
R1
R2
R3
R4
C1
C2
C3
C4
PB4
PB5
PB9
PA8
PB8
PA9
PB0
PB2


PB1
PB3
PA12
PB7
PB6
PA0
1
A
F
2
3
B
A
F
B
G
E
C
D
dot
E
D
dot
C
G
4
PA5
PA4
PA6
PA1
PA11
PA7

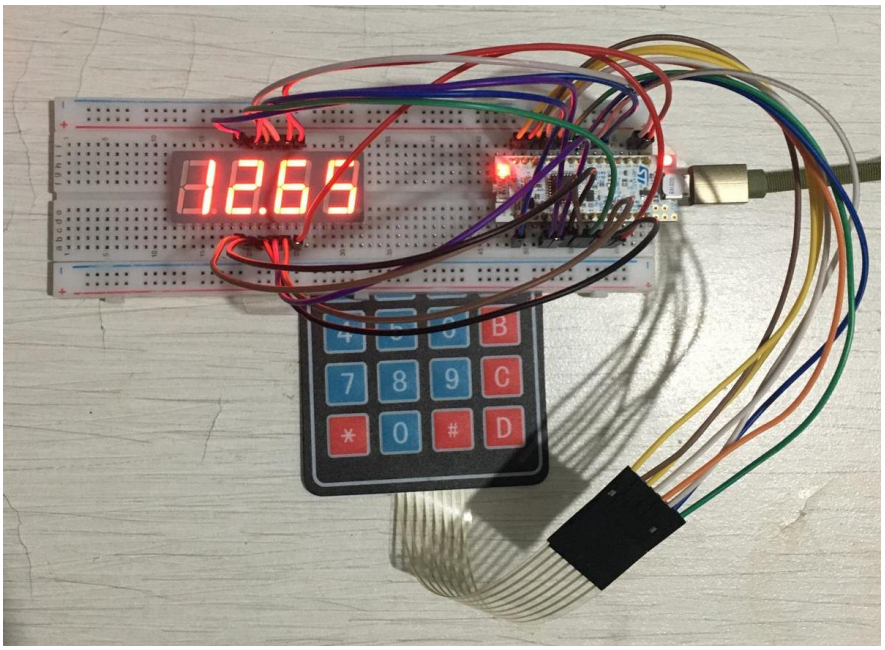
4. FLOWCHART

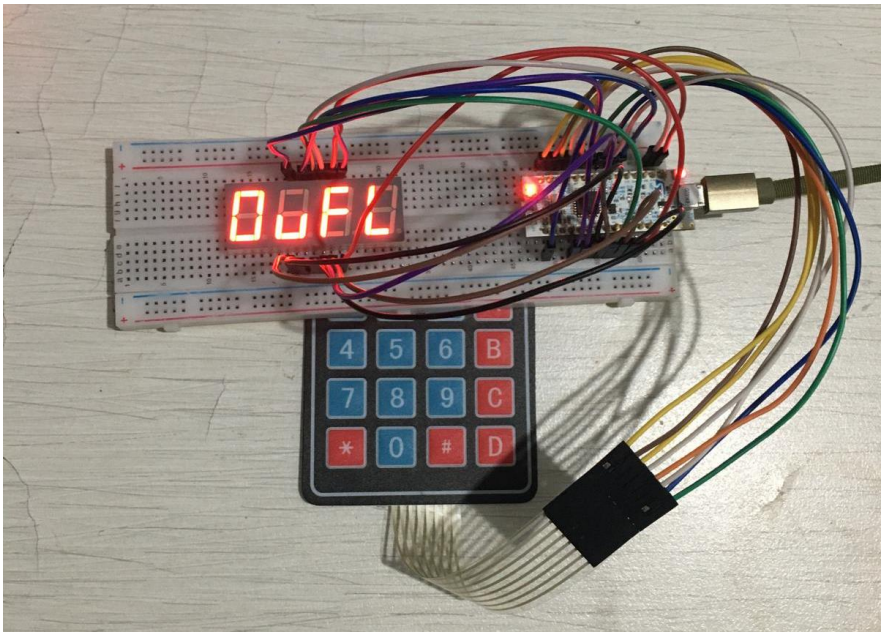


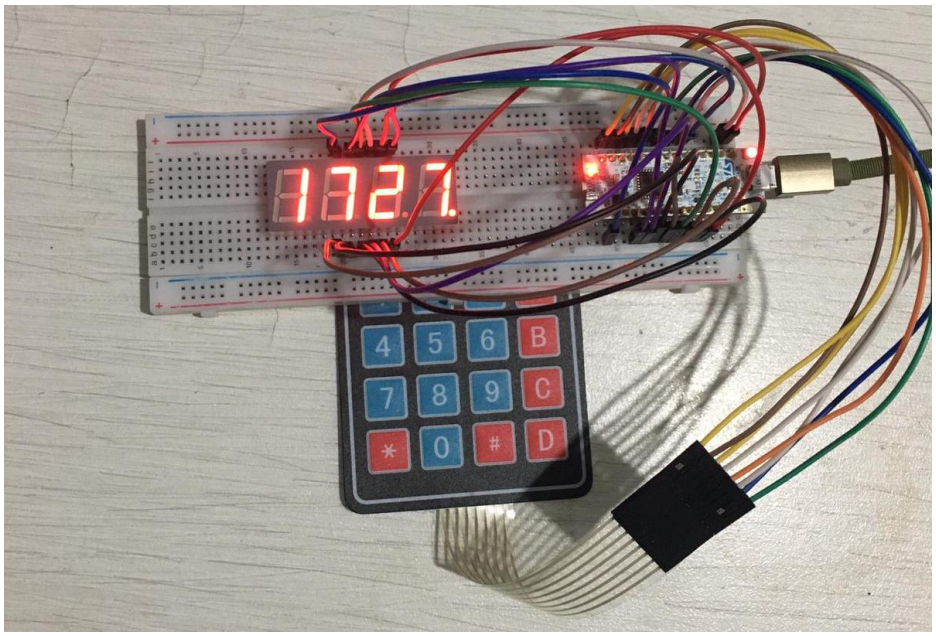
5. TASK

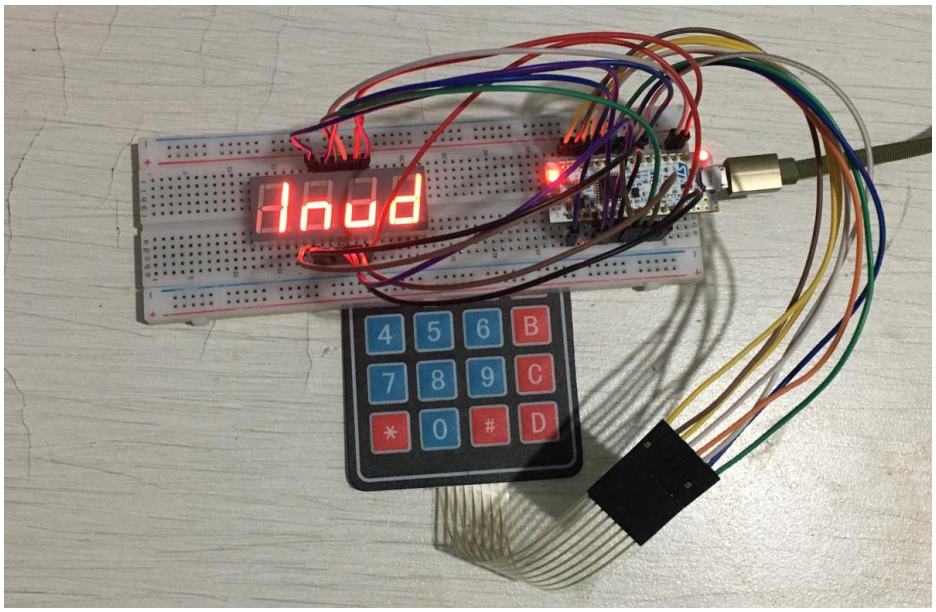
- nucleo-32 card schematıcs examıned
- The seven segment display block diagram was examined
- Created flowchart
- Created blockdiagram
- Necessary connections are made on the breadboard
- Keypad and Seven Segment Display connection has been made
- Determine interrupts
- Utilize interrupts for all functionality
- School ID shows by **displayID_SSD** func
- When keys are entered ssd shows the numbers
- No more than 4 digits by **Keypad_data** func
- ABCDEF keys use as by “calc.c”
- Floating point number system by **utility_SSD** func
- Negative numbers have negative sign by **utility_SSD** func
- If the result overflows display shows OuFL by **overflow_SSD** func
- If the operation is invalid display shows Invd by **invalid_SSD** func

6. PROJECT SETUP WITH PICTURES:









7. PARTS LIST:

LIST		
MATERIAL	PIECE	PRICE
BOARD	1	7
7-SEGMENT (1X4)	1	9
CABLE	1	5
4X4 KEYPAD	1	9
STM32	1	95

8. CONCLUSION:

The subject of this project is to create a fully operational scientific calculator in C. Before starting to write the code, a flowchart was created and a blockdiagram was drawn so that the flowchart was used while writing the code. As for the code part. A keypad and a seven-segment display are attached. On power up SSD show my school ID number (first 2 and last 2 digits). When number pressed, everything is cleared and only the number which taken from keypad display. After each key entered, the number is shifted to the left and displayed on the SSD screen. If the digits are already full, new number key presses are ignored by func. ABCDE keys call the calculation functions assigned to them. Negative numbers have negative sign. If the numbers overflows SSD display OuFL. If the operation is invalid SSD display Invd. The problems we encountered in this project are; button bouncing takes value 2 or 3 times when we press the button until we take our hand. As a result, I was able to create a scientific calculator at the end of the project.

9. REFERANCES:

- https://www.instructables.com/Using-a-4-digit-7-segment-display-with-arduino/
- https://en.wikipedia.org/wiki/Seven-segment_display
- https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet
- http://wiki.sunfounder.cc/index.php?title=4X4_Matrix_Keypad_Module

Appendix Codes:

```c
/*
 * keypad.h
 *
 *  Created on: Dec 19, 2020
 *      Author: Mehmet Akif/171024027
 */

#ifndef KEYPAD_H_
#define KEYPAD_H_

#include "calc.h"
#include"main.h"

/*Keypad related function*/
void keypad_init();     //initiation for keypad pins
void clear_rows_keypad(); // set 0  keypad rows
void set_rows_keypad(); // set 1 keypad rows


/* taken data from button which is pressed
 and figure out which button is this*/
void Keypad_data(uint8_t a);


#endif /* KEYPAD_H_ */

```

```c
/*
 * keypad.c
 *
 *  Created on: Dec 19, 2020
 *      Author: Mehmet Akif/171024027
 *      description:In this section, necessary pins of
 *      the keypad have been activated in order for the keypad
 *      buttons to receive data. Next, the interrupt was created
 *      for the buttons. Thanks to this interrupt, when the button
 *      is pressed, it is processed according to priority.
 *      After determining which character the received data is from,
 *      it was sent to the display function for printing. It was sent to
 *      the calculation function for the necessary operations.
 */
#include "keypad.h"

/*to the reach delay func*/
extern void delay_ms(volatile unsigned int);

void keypad_init(void){

       /* Enable GPIOB and GPIOA clock */
        RCC->IOPENR |= (1U << 1);
        RCC->IOPENR |= (1U << 0);


        /* Setup PA8,PB9,PB5 and PB4 as output (rows)*/
        GPIOA->MODER &= ~(3U << 2*8);
        GPIOA->MODER |= (1U << 2*8);//PA8 is output

        GPIOB->MODER &= ~(3U << 2*9);
        GPIOB->MODER |= (1U << 2*9);//PB9 is output

        GPIOB->MODER &= ~(3U << 2*5);
        GPIOB->MODER |= (1U << 2*5);//PB5 is output

        GPIOB->MODER &= ~(3U << 2*4);
        GPIOB->MODER |= (1U << 2*4);//PB4 is output



        /* Setup PA9,PB0,PB2 and PB8  as input(colums) */
        GPIOA->MODER &= ~(3U << 2*9);// PA9 is input
        GPIOA->PUPDR |= (2U << 2*9); // Pull-down mode

        GPIOB->MODER &= ~(3U << 2*0);//PB0 is input
        GPIOB->PUPDR |= (2U << 2*0); // Pull-down mode

        GPIOB->MODER &= ~(3U << 2*2);//PB2 is input
        GPIOB->PUPDR |= (2U << 2*2); // Pull-down mode

        GPIOB->MODER &= ~(3U << 2*8);//PB8 is input
        GPIOB->PUPDR |= (2U << 2*8); // Pull-down mode


        /*setup interrupts for inputs*/
        EXTI->EXTICR[2] |=(0U << 8*1);//PA9
        EXTI->EXTICR[0] |=(1U << 0);//PB0
        EXTI->EXTICR[0] |=(1U << 2*8);//PB2
        EXTI->EXTICR[2] |=(1U << 0);//PB8


```

```
63          /*rising edge*/
64          EXTI->RTSR1 |= (1U << 9);// 9th pin
65          EXTI->RTSR1 |= (1U << 0);// 0th pin
66          EXTI->RTSR1 |= (1U << 2);// 2th pin
67          EXTI->RTSR1 |= (1U << 8);// 8th pin
68
69
70          /* MASK*/
71          EXTI->IMR1 |= (1U << 9);// 9th pin
72          EXTI->IMR1 |= (1U << 0);// 0th pin
73          EXTI->IMR1 |= (1U << 2);// 2th pin
74          EXTI->IMR1 |= (1U << 8);// 8th pin
75
76
77          /*NVIC*/
78          NVIC_SetPriority(EXTI0_1_IRQn,0);
79          NVIC_EnableIRQ(EXTI0_1_IRQn);
80
81          NVIC_SetPriority(EXTI2_3_IRQn,0);
82          NVIC_EnableIRQ(EXTI2_3_IRQn);
83
84          NVIC_SetPriority(EXTI4_15_IRQn,0);
85          NVIC_EnableIRQ(EXTI4_15_IRQn);
86
87  }
88  /* interrut for PB0*/
89  void EXTI0_1_IRQHandler(void){
90      if (EXTI->RPR1 & (1U << 0)){// check if pending register equal 1
91
92          clear_rows_keypad();
93          /* make PA8 enable*/
94          GPIOA->ODR ^=( 1U << 8);
95          if ((GPIOB->IDR >> 0) & 1){//check if PB0 equal 1
96              /* #=(F) character*/
97              Keypad_data(15);
98
99          }
100         /*make PA8 disable*/
101         GPIOA->ODR ^=( 1U << 8); // PA8
102
103         /* make PB9 enable*/
104         GPIOB->ODR ^=( 1U << 9); // PB9
105         if ((GPIOB->IDR >> 0) & 1){
106             /* 9 character*/
107             Keypad_data(9);
108
109         }
110         /* make PB9 disable*/
111         GPIOB->ODR ^=( 1U << 9); // PB9
112
113         /* make PB5 enable*/
114         GPIOB->ODR ^=( 1U << 5); // PB5
115         if ((GPIOB->IDR >> 0) & 1){
116             /* 6 character*/
117             Keypad_data(6);
118
119         }
120         /* make PB5 disable*/
121         GPIOB->ODR ^=( 1U << 5); // PB5
122
123         /* make PB4 enable*/
124         GPIOB->ODR ^=( 1U << 4); // PB4
```

```
125         if ((GPIOB->IDR >> 0) & 1){
126             /* 3 character*/
127             Keypad_data(3);
128 
129         }
130         /* make PB4 disable*/
131         GPIOB->ODR ^=( 1U << 4); // PB4
132 
133 
134         set_rows_keypad();
135         /*clear interrupt for clear pending register */
136         EXTI->RPR1 |= (1U << 0);
137     }
138 }
139 
140 /* interrut for PB2*/
141 void EXTI2_3_IRQHandler(void){
142 
143     if (EXTI->RPR1 & (1U << 2)){// check if pending register equal 1
144 
145         clear_rows_keypad();
146         /*make PA8 enable*/
147         GPIOA->ODR ^=( 1U << 8); // PA8
148         if ((GPIOB ->IDR >> 2) & 1){//check if PB2 equal 1
149             /* D character*/
150             Keypad_data(13);
151 
152         }
153         /*make PA8 disable*/
154         GPIOA->ODR ^=( 1U << 8); // PA8
155 
156         /* make PB9 enable*/
157         GPIOB->ODR ^=( 1U << 9); // PB9
158         if ((GPIOB ->IDR >> 2) & 1){
159             /* C character*/
160             Keypad_data(12);
161 
162         }
163         /* make PB9 disable*/
164         GPIOB->ODR ^=( 1U << 9); // PB9
165 
166         /* make PB5 enable*/
167         GPIOB->ODR ^=( 1U << 5); // PB5
168         if ((GPIOB ->IDR >> 2) & 1){
169             /* B character*/
170             Keypad_data(11);
171 
172         }
173         /* make PB5 disable*/
174         GPIOB->ODR ^=( 1U << 5); // PB5
175 
176         /* make PB4 enable*/
177         GPIOB->ODR ^=( 1U << 4); // PB4
178         if ((GPIOB ->IDR >> 2) & 1){
179             /* A character*/
180             Keypad_data(10);
181 
182         }
183         /* make PB4 disable*/
184         GPIOB->ODR ^=( 1U << 4); // PB4
185 
186 
```

```
187         set_rows_keypad();
188         /*clear interrupt for clear pending register */
189         EXTI->RPR1 |= (1U << 2);
190     }
191 }
192
193 /* interrut for PB8 and PA9*/
194 void EXTI4_15_IRQHandler(void){
195
196             /*interrut for PB8*/
197         if (EXTI->RPR1 & (1U << 8)){// check if pending register equal 1
198             clear_rows_keypad();
199             /*make PA8 enable*/
200             GPIOA->ODR ^=( 1U << 8); // PA8
201             if ((GPIOB ->IDR >> 8) & 1){//check if PB8 equal 1
202                 /* *(E) character*/
203                 Keypad_data(14);
204
205             }
206             /*make PA8 disable*/
207             GPIOA->ODR ^=( 1U << 8); // PA8
208
209             /* make PB9 enable*/
210             GPIOB->ODR ^=( 1U << 9); // PB9
211             if ((GPIOB ->IDR >> 8) & 1){
212                 /* 7 character*/
213                 Keypad_data(7);
214
215             }
216             /* make PB9 disable*/
217             GPIOB->ODR ^=( 1U << 9); // PB9
218
219             /* make PB5 enable*/
220             GPIOB->ODR ^=( 1U << 5); // PB5
221             if ((GPIOB ->IDR >> 8) & 1){
222                 /* 4 character*/
223                 Keypad_data(4);
224
225             }
226             /* make PB5 disable*/
227             GPIOB->ODR ^=( 1U << 5); // PB5
228
229             /* make PB4 enable*/
230             GPIOB->ODR ^=( 1U << 4); // PB4
231             if ((GPIOB ->IDR >> 8) & 1){
232                 /* 1 character*/
233                 Keypad_data(1);
234
235             }
236             /* make PB4 disable*/
237             GPIOB->ODR ^=( 1U << 4); // PB4
238
239
240             set_rows_keypad();
241             /*clear interrupt for clear pending register */
242             EXTI->RPR1 |= (1U << 8);
243         }
244
245         /*interrut for PA9*/
246         if (EXTI->RPR1 & (1U << 9)){// check if pending register equal 1
247             clear_rows_keypad();
248             /*make PA8 enable*/
```

```
249             GPIOA->ODR ^=( 1U << 8); //check if PA8 equal 1
250             if ((GPIOA ->IDR >> 9) & 1){
251                 /* 0 character*/
252                 Keypad_data(0);
253 
254             }
255             /*make PA8 disable*/
256             GPIOA->ODR ^=( 1U << 8); // PA8
257 
258             /* make PB9 enable*/
259             GPIOB->ODR ^=( 1U << 9); // PB9
260             if ((GPIOA ->IDR >> 9) & 1){
261                 /* 8 character*/
262                 Keypad_data(8);
263 
264             }
265             /* make PB9 disable*/
266             GPIOB->ODR ^=( 1U << 9); // PB9
267 
268             /* make PB5 enable*/
269             GPIOB->ODR ^=( 1U << 5); // PB5
270             if ((GPIOA ->IDR >> 9) & 1){
271                 /* 5 character*/
272                 Keypad_data(5);
273 
274             }
275             /* make PB5 disable*/
276             GPIOB->ODR ^=( 1U << 5); // PB5
277 
278             /* make PB4 enable*/
279             GPIOB->ODR ^=( 1U << 4); // PB4
280             if ((GPIOA ->IDR >> 9) & 1){
281                 /* 2 character*/
282                 Keypad_data(2);
283 
284             }
285             /* make PB4 disable*/
286             GPIOB->ODR ^=( 1U << 4); // PB4
287 
288 
289             set_rows_keypad();
290 
291             /*clear interrupt for clear pending register */
292             EXTI->RPR1 |= (1U << 9);
293         }
294 
295 }
296 
297 
298 void clear_rows_keypad(void){
299         /*clearing the rows here*/
300         GPIOA->ODR &= ~(1U << 8);//PA8
301         GPIOB->ODR &= ~(1U << 9);//PB9
302         GPIOB->ODR &= ~(1U << 5);//PB5
303         GPIOB->ODR &= ~(1U << 4);//PB4
304 }
305 
306 void set_rows_keypad(void){
307         /*seting the rows here*/
308         GPIOA->ODR |= (1U << 8);//PA8
309         GPIOB->ODR |= (1U << 9);//PB9
310         GPIOB->ODR |= (1U << 5);//PB5
```

```
311         GPIOB->ODR |= (1U << 4);//PB4
312
313 }
314
315 void Keypad_data(uint8_t a){
316
317     static int i = 0;
318
319     /*
320     *if the digits are already full,
321     *new number key presses are ignored
322     *by counter int i.
323     **/
324     if ((a<10) & (i<4)){
325         calculation.x = (calculation.x * 10) + (float)a;
326         utility_SSD(calculation.x);
327         i++;
328     }
329     /*
330     *if ABCDE pressed,
331     *calculation mode will be selected
332     **/
333     else if ((a>9) & (i<=4)){
334         i = 4;
335
336         if(a == 10){
337             calculation.current_process = Addition;
338             displaychar_SSD(Addition);
339             i = 7;
340         }
341         else if(a == 11){
342             calculation.current_process = Substraction;
343             displaychar_SSD(Substraction);
344             i = 7;
345         }
346         else if(a == 12){
347             calculation.current_process = Multiplacation;
348             displaychar_SSD(Multiplacation);
349             i = 7;
350         }
351         else if(a == 13){
352             calculation.current_process = Division;
353             displaychar_SSD(Division);
354             i = 7;
355         }
356         else if(a == 14){
357             calculation.current_process = E;
358             displaychar_SSD(E);
359             i = 5;
360         }
361     }
362 /*
363  * if E Key is pressed, scientific mode on
364  * and expect another keypress
365  */
366     else if (i == 5){
367
368         if(a == 10){
369             calculation.current_process = Log;
370             displaychar_SSD(Log);
371             i = 11;
372         }
```

```
373         else if(a == 11){
374             calculation.current_process = Ln;
375             displaychar_SSD(Ln);
376             i = 11;
377         }
378         else if(a == 12){
379             calculation.current_process = Sqrt;
380             displaychar_SSD(Sqrt);
381             i = 11;
382         }
383         else if(a == 13){
384             calculation.current_process = Pow2;
385             displaychar_SSD(Pow2);
386             i = 11;
387         }
388         else if(a == 14){
389             calculation.current_process = EE;
390             displaychar_SSD(EE);
391             i = 6;
392         }
393     }
394     /*
395      * if EE Key is pressed, trigonometric mode on
396      * and expect another keypress
397      */
398     else if (i == 6){
399 
400         if(a == 10){
401             calculation.current_process = Sin;
402             displaychar_SSD(Sin);
403             i = 11;
404         }
405         else if(a == 11){
406             calculation.current_process = Cos;
407             displaychar_SSD(Cos);
408             i = 11;
409         }
410         else if(a == 12){
411             calculation.current_process = Tan;
412             displaychar_SSD(Tan);
413             i = 11;
414         }
415         else if(a == 13){
416             calculation.current_process = Cot;
417             displaychar_SSD(Cot);
418             i = 11;
419         }
420         else if(a == 14){
421             calculation.x = 3.141;
422             utility_SSD(calculation.x);
423             i = 4;
424         }
425     }
426     /*
427      * if ABCD pressed, requires another number
428      * for calculation
429      */
430     else if((a<10) & (i >= 7) & (i < 11)){
431         calculation.y = (calculation.y * 10) + (float)a;
432         utility_SSD(calculation.y);
433         i++;
434     }
```

```
435     /*
436      * F key is for equal
437      */
438     else if ((i == 11) | (a == 15)){
439 
440         calculate();
441         i = 0;
442         calculation.x = calculation.result;
443         calculation.y = 0;
444         calculation.current_process = 0;
445         utility_SSD(calculation.x);
446     }
447 }
448 
449 
450 
```

```c
/*
 * display.h
 *
 *  Created on: Dec 19, 2020
 *      Author: Mehmet Akif/171024027
 */

#ifndef DISPLAY_H_
#define DISPLAY_H_

#include "main.h"
#include "bsp.h"

typedef struct{
    uint8_t Digits[4];
    uint8_t Oflw:1;
    uint8_t sign:1;
    uint8_t dot:3;
    uint8_t Inv:1;
}SSD;

/*
 * Display struct keep the digits and
 * overflow, sign, dot, invalid bits
 */
SSD Display;

/*
 * initiation for keypad pins
 */
void init_SSD();

/*
 * This function ensures that the digits on the display
 * are lit by quickly flashing the digits.
 */
void display_SSD();

/*
 * the cases which are inside of this func show that
 * how to display the character
 */
void printDigit_SSD(uint8_t);

void displayID_SSD();

/*
 * when the result bigger than 9999 or less than -999
 * display shows that OuFL
 */
void overflow_SSD();

/*
 * when the operation is invalid i.e. 3/0 or sqrt(-2))
 * display shows that Invd
 */
void invalid_SSD();

/*
 * separates the incoming result into digit
 * and we can see that if number is negative or
 * not through sign bit
```

```
 */
void utility_SSD(float var);

/*
 * It determines which character should be lit on which
 * digit by assigning case values to digit.
 */
void displaychar_SSD(uint8_t x);

#endif /* DISPLAY_H_ */

```

```c
/*
 * display.c
 *
 *  Created on: Dec 19, 2020
 *      Author: Mehmet Akif/171024027
 *
 *      description:In this section, necessary pins of
 *      the SSD have been activated in order for the lid ssd.
 *      Next,With the display function, the burning of the digits
 *      at the same time is provided by flashing the digits invisibly.
 *      and then we display the character from printdigit func
 */

#include "display.h"

extern void main();



void init_SSD(){
    GPIOB->MODER &= ~(3U << 2*1);
    GPIOB->MODER |= (1U << 2*1);//PB1 is output

    GPIOB->MODER &= ~(3U << 2*3);
    GPIOB->MODER |= (1U << 2*3);//PB3 is output

    GPIOB->MODER &= ~(3U << 2*6);
    GPIOB->MODER |= (1U << 2*6);//PB6 is output

    GPIOB->MODER &= ~(3U << 2*7);
    GPIOB->MODER |= (1U << 2*7);//PB7 is output

    GPIOA->MODER &= ~(3U << 2*0);
    GPIOA->MODER |= (1U << 2*0);//PA0 is output

    GPIOA->MODER &= ~(3U << 2*1);
    GPIOA->MODER |= (1U << 2*1);//PA1 is output

    GPIOA->MODER &= ~(3U << 2*4);
    GPIOA->MODER |= (1U << 2*4);//PA4 is output

    GPIOA->MODER &= ~(3U << 2*5);
    GPIOA->MODER |= (1U << 2*5);//PA5 is output

    GPIOA->MODER &= ~(3U << 2*6);
    GPIOA->MODER |= (1U << 2*6);//PA6 is output

    GPIOA->MODER &= ~(3U << 2*7);
    GPIOA->MODER |= (1U << 2*7);//PA7 is output

    GPIOA->MODER &= ~(3U << 2*11);
    GPIOA->MODER |= (1U << 2*11);//PA11 is output

    GPIOA->MODER &= ~(3U << 2*12);
    GPIOA->MODER |= (1U << 2*12);//PA12 is output


}

void display_SSD(){

    static int i = 0;
```

```
63          if(Display.Inv == 1){
64              invalid_SSD();
65
66          }
67          else if(Display.Oflw == 1){
68              overflow_SSD();
69
70          }
71
72          if(i == 1){
73              GPIOA->ODR |= (1U << 7); //PA7
74              GPIOB->ODR &= ~(1U << 6); //PB6
75              GPIOB->ODR &= ~(1U << 7); //PB7
76              GPIOB->ODR &= ~(1U << 1); //PB1
77              printDigit_SSD(Display.Digits[0]);
78              GPIOA->ODR |= ( 1U << 6); // PA6
79              if(Display.dot == 0) GPIOA->ODR &= ~( 1U << 6);
80          }
81          else if(i == 10){
82              GPIOA->ODR &= ~(1U << 7); //PA7
83              GPIOB->ODR |= (1U << 6); //PB6
84              GPIOB->ODR &= ~(1U << 7); //PB7
85              GPIOB->ODR &= ~(1U << 1); //PB1
86              printDigit_SSD(Display.Digits[1]);
87              GPIOA->ODR |= ( 1U << 6); // PA6
88              if(Display.dot == 1) GPIOA->ODR &= ~( 1U << 6);
89          }
90
91          else if(i == 20){
92              GPIOA->ODR &= ~(1U << 7); //PA7
93              GPIOB->ODR &= ~(1U << 6); //PB6
94              GPIOB->ODR |= (1U << 7); //PB7
95              GPIOB->ODR &= ~(1U << 1); //PB1
96              printDigit_SSD(Display.Digits[2]);
97              GPIOA->ODR |= ( 1U << 6); // PA6
98              if(Display.dot == 2) GPIOA->ODR &= ~( 1U << 6);
99          }
100         else if(i == 30){
101             GPIOA->ODR &= ~(1U << 7); //PA7
102             GPIOB->ODR &= ~(1U << 6); //PB6
103             GPIOB->ODR &= ~(1U << 7); //PB7
104             GPIOB->ODR |= (1U << 1); //PB1
105             printDigit_SSD(Display.Digits[3]);
106             GPIOA->ODR |= ( 1U << 6); // PA6
107             if(Display.dot == 3) GPIOA->ODR &= ~( 1U << 6);
108         }
109         else if(i == 40) i = 0;
110
111         i++;
112
113
114
115
116 }
117
118 void printDigit_SSD(uint8_t x){
119
120         switch(x){
121         case 0:     //0
122
123             GPIOB->ODR &= ~( 1U << 3); // PB3
124             GPIOA->ODR &= ~( 1U << 0); // PA0
```

```
125             GPIOA->ODR &= ~( 1U << 1); // PA1
126             GPIOA->ODR &= ~( 1U << 4); // PA4
127             GPIOA->ODR &= ~( 1U << 5); // PA5
128             GPIOA->ODR &= ~( 1U << 12); // PA12
129             GPIOA->ODR |=  ( 1U << 11); // PA11
130 
131             break;
132 
133         case 1:     //1
134             GPIOB->ODR |= ( 1U << 3); // PB3
135             GPIOA->ODR &= ~( 1U << 0); // PA0
136             GPIOA->ODR &= ~( 1U << 1); // PA1
137             GPIOA->ODR |= ( 1U << 4); // PA4
138             GPIOA->ODR |= ( 1U << 5); // PA5
139             GPIOA->ODR |= ( 1U << 12); // PA12
140             GPIOA->ODR |= ( 1U << 11); // PA11
141 
142             break;
143 
144         case 2:     //2
145             GPIOB->ODR &= ~( 1U << 3); // PB3
146             GPIOA->ODR &= ~( 1U << 0); // PA0
147             GPIOA->ODR |=  ( 1U << 1); // PA1
148             GPIOA->ODR &= ~( 1U << 4); // PA4
149             GPIOA->ODR &= ~( 1U << 5); // PA5
150             GPIOA->ODR |=  ( 1U << 12); // PA12
151             GPIOA->ODR &= ~( 1U << 11); // PA11
152 
153             break;
154 
155         case 3:     //3
156 
157             GPIOB->ODR &= ~( 1U << 3); // PB3
158             GPIOA->ODR &= ~( 1U << 0); // PA0
159             GPIOA->ODR &= ~( 1U << 1); // PA1
160             GPIOA->ODR &= ~( 1U << 4); // PA4
161             GPIOA->ODR |=  ( 1U << 5); // PA5
162             GPIOA->ODR |=  ( 1U << 12); // PA12
163             GPIOA->ODR &= ~( 1U << 11); // PA11
164 
165             break;
166 
167         case 4:     //4
168             GPIOB->ODR |=  ( 1U << 3); // PB3
169             GPIOA->ODR &= ~( 1U << 0); // PA0
170             GPIOA->ODR &= ~( 1U << 1); // PA1
171             GPIOA->ODR |=  ( 1U << 4); // PA4
172             GPIOA->ODR |=  ( 1U << 5); // PA5
173             GPIOA->ODR &= ~( 1U << 12); // PA12
174             GPIOA->ODR &= ~( 1U << 11); // PA11
175 
176             break;
177 
178         case 5:     //5
179 
180             GPIOB->ODR &= ~( 1U << 3); // PB3
181             GPIOA->ODR |=  ( 1U << 0); // PA0
182             GPIOA->ODR &= ~( 1U << 1); // PA1
183             GPIOA->ODR &= ~( 1U << 4); // PA4
184             GPIOA->ODR |=  ( 1U << 5); // PA5
185             GPIOA->ODR &= ~( 1U << 12); // PA12
186             GPIOA->ODR &= ~( 1U << 11); // PA11
```

```
187
188             break;
189
190         case 6:     //6
191             GPIOB->ODR &= ~( 1U << 3); // PB3
192             GPIOA->ODR |=  ( 1U << 0); // PA0
193             GPIOA->ODR &= ~( 1U << 1); // PA1
194             GPIOA->ODR &= ~( 1U << 4); // PA4
195             GPIOA->ODR &= ~( 1U << 5); // PA5
196             GPIOA->ODR &= ~( 1U << 12); // PA12
197             GPIOA->ODR &= ~( 1U << 11); // PA11
198
199             break;
200
201         case 7:     //7
202
203             GPIOB->ODR &= ~( 1U << 3); // PB3
204             GPIOA->ODR &= ~( 1U << 0); // PA0
205             GPIOA->ODR &= ~( 1U << 1); // PA1
206             GPIOA->ODR |=  ( 1U << 4); // PA4
207             GPIOA->ODR |=  ( 1U << 5); // PA5
208             GPIOA->ODR |=  ( 1U << 12); // PA12
209             GPIOA->ODR |=  ( 1U << 11); // PA11
210
211             break;
212
213         case 8:     //8
214
215             GPIOB->ODR &= ~( 1U << 3); // PB3
216             GPIOA->ODR &= ~( 1U << 0); // PA0
217             GPIOA->ODR &= ~( 1U << 1); // PA1
218             GPIOA->ODR &= ~( 1U << 4); // PA4
219             GPIOA->ODR &= ~( 1U << 5); // PA5
220             GPIOA->ODR &= ~( 1U << 12); // PA12
221             GPIOA->ODR &= ~( 1U << 11); // PA11
222
223             break;
224
225         case 9:     //9
226             GPIOB->ODR &= ~( 1U << 3); // PB3
227             GPIOA->ODR &= ~( 1U << 0); // PA0
228             GPIOA->ODR &= ~( 1U << 1); // PA1
229             GPIOA->ODR &= ~( 1U << 4); // PA4
230             GPIOA->ODR |=  ( 1U << 5); // PA5
231             GPIOA->ODR &= ~( 1U << 12); // PA12
232             GPIOA->ODR &= ~( 1U << 11); // PA11
233
234             break;
235
236         case 10://A
237
238             GPIOB->ODR &= ~( 1U << 3); // PB3
239             GPIOA->ODR &= ~( 1U << 0); // PA0
240             GPIOA->ODR &= ~( 1U << 1); // PA1
241             GPIOA->ODR &= ~( 1U << 4); // PA4
242             GPIOA->ODR &= ~( 1U << 5); // PA5
243             GPIOA->ODR |=  ( 1U << 12); // PA12
244             GPIOA->ODR &= ~( 1U << 11); // PA11
245
246
247
248             break;
```

```
249
250         case 11://B
251
252             GPIOB->ODR |=  ( 1U << 3); // PB3
253             GPIOA->ODR |=  ( 1U << 0); // PA0
254             GPIOA->ODR &= ~( 1U << 1); // PA1
255             GPIOA->ODR &= ~( 1U << 4); // PA4
256             GPIOA->ODR &= ~( 1U << 5); // PA5
257             GPIOA->ODR &= ~( 1U << 12); // PA12
258             GPIOA->ODR &= ~( 1U << 11); // PA11
259
260
261             break;
262
263         case 12://C
264             GPIOB->ODR &= ~( 1U << 3); // PB3
265             GPIOA->ODR |=  ( 1U << 0); // PA0
266             GPIOA->ODR |=  ( 1U << 1); // PA1
267             GPIOA->ODR &= ~( 1U << 4); // PA4
268             GPIOA->ODR &= ~( 1U << 5); // PA5
269             GPIOA->ODR &= ~( 1U << 12); // PA12
270             GPIOA->ODR |=  ( 1U << 11); // PA11
271
272             break;
273
274         case 13://D
275             GPIOB->ODR |=  ( 1U << 3); // PB3
276             GPIOA->ODR &= ~( 1U << 0); // PA0
277             GPIOA->ODR &= ~( 1U << 1); // PA1
278             GPIOA->ODR &= ~( 1U << 4); // PA4
279             GPIOA->ODR &= ~( 1U << 5); // PA5
280             GPIOA->ODR |=  ( 1U << 12); // PA12
281             GPIOA->ODR &= ~( 1U << 11); // PA11
282
283             break;
284
285         case 14://E
286             GPIOB->ODR &= ~( 1U << 3); // PB3
287             GPIOA->ODR |=  ( 1U << 0); // PA0
288             GPIOA->ODR |=  ( 1U << 1); // PA1
289             GPIOA->ODR &= ~( 1U << 4); // PA4
290             GPIOA->ODR &= ~( 1U << 5); // PA5
291             GPIOA->ODR &= ~( 1U << 12); // PA12
292             GPIOA->ODR &= ~( 1U << 11); // PA11
293
294
295             break;
296
297         case 15: //F
298             GPIOB->ODR &= ~( 1U << 3); // PB3
299             GPIOA->ODR |=  ( 1U << 0); // PA0
300             GPIOA->ODR |=  ( 1U << 1); // PA1
301             GPIOA->ODR |=  ( 1U << 4); // PA4
302             GPIOA->ODR &= ~( 1U << 5); // PA5
303             GPIOA->ODR &= ~( 1U << 12); // PA12
304             GPIOA->ODR &= ~( 1U << 11); // PA11
305             break;
306
307
308         case 30: //u
309             GPIOB->ODR |=  ( 1U << 3); // PB3
310             GPIOA->ODR |=  ( 1U << 0); // PA0
```

```
311             GPIOA->ODR &= ~( 1U << 1); // PA1
312             GPIOA->ODR &= ~( 1U << 4); // PA4
313             GPIOA->ODR &= ~( 1U << 5); // PA5
314             GPIOA->ODR |=  ( 1U << 12); // PA12
315             GPIOA->ODR |=  ( 1U << 11); // PA11
316             break;
317 
318         case 31: //L
319             GPIOB->ODR |=  ( 1U << 3); // PB3
320             GPIOA->ODR |=  ( 1U << 0); // PA0
321             GPIOA->ODR |=  ( 1U << 1); // PA1
322             GPIOA->ODR &= ~( 1U << 4); // PA4
323             GPIOA->ODR &= ~( 1U << 5); // PA5
324             GPIOA->ODR &= ~( 1U << 12); // PA12
325             GPIOA->ODR |=  ( 1U << 11); // PA11
326             break;
327 
328         case 32: //n
329             GPIOB->ODR |=  ( 1U << 3); // PB3
330             GPIOA->ODR |=  ( 1U << 0); // PA0
331             GPIOA->ODR &= ~( 1U << 1); // PA1
332             GPIOA->ODR |=  ( 1U << 4); // PA4
333             GPIOA->ODR &= ~( 1U << 5); // PA5
334             GPIOA->ODR |=  ( 1U << 12); // PA12
335             GPIOA->ODR &= ~( 1U << 11); // PA11
336             break;
337 
338         case 33: //D
339             GPIOB->ODR |=  ( 1U << 3); // PB3
340             GPIOA->ODR &= ~( 1U << 0); // PA0
341             GPIOA->ODR &= ~( 1U << 1); // PA1
342             GPIOA->ODR &= ~( 1U << 4); // PA4
343             GPIOA->ODR &= ~( 1U << 5); // PA5
344             GPIOA->ODR |=  ( 1U << 12); // PA12
345             GPIOA->ODR &= ~( 1U << 11); // PA11
346             break;
347 
348         case 34: // negative sign
349             GPIOB->ODR |= ( 1U << 3); // PB3
350             GPIOA->ODR |= ( 1U << 0); // PA0
351             GPIOA->ODR |= ( 1U << 1); // PA1
352             GPIOA->ODR |= ( 1U << 4); // PA4
353             GPIOA->ODR |= ( 1U << 5); // PA5
354             GPIOA->ODR |= ( 1U << 12); // PA12
355             GPIOA->ODR &= ~( 1U << 11); // PA11
356             break;
357 
358         case 35: // space
359             GPIOB->ODR |= ( 1U << 3); // PB3
360             GPIOA->ODR |= ( 1U << 0); // PA0
361             GPIOA->ODR |= ( 1U << 1); // PA1
362             GPIOA->ODR |= ( 1U << 4); // PA4
363             GPIOA->ODR |= ( 1U << 5); // PA5
364             GPIOA->ODR |= ( 1U << 12); // PA12
365             GPIOA->ODR |= ( 1U << 11); // PA11
366             break;
367         }
368 }
369 
370 
371 void displaychar_SSD(uint8_t x){
372     Display.dot = 5;
```

```
373     switch(x){
374     case 0: //-
375         Display.Digits[0] = 34;
376         Display.Digits[1] = 35;
377         Display.Digits[2] = 35;
378         Display.Digits[3] = 35;
379         break;
380     case 1: //'A'
381         Display.Digits[3] = 10;
382         Display.Digits[0] = 35;
383         Display.Digits[2] = 35;
384         Display.Digits[1] = 35;
385         break;
386     case 2: //'B'
387         Display.Digits[3] = 11;
388         Display.Digits[1] = 35;
389         Display.Digits[2] = 35;
390         Display.Digits[0] = 35;
391         break;
392     case 3:  //'C'
393         Display.Digits[3] = 12;
394         Display.Digits[1] = 35;
395         Display.Digits[2] = 35;
396         Display.Digits[0] = 35;
397         break;
398     case 4: //'D'
399         Display.Digits[3] = 13;
400         Display.Digits[1] = 35;
401         Display.Digits[2] = 35;
402         Display.Digits[0] = 35;
403         break;
404     case 5:  //'E'
405         Display.Digits[3] = 14;
406         Display.Digits[2] = 35;
407         Display.Digits[1] = 35;
408         Display.Digits[0] = 35;
409         break;
410     case 6: //'EA'
411         Display.Digits[3] = 14;
412         Display.Digits[2] = 10;
413         Display.Digits[1] = 35;
414         Display.Digits[0] = 35;
415         break;
416     case 7: //'EB'
417         Display.Digits[3] = 14;
418         Display.Digits[2] = 11;
419         Display.Digits[1] = 35;
420         Display.Digits[0] = 35;
421         break;
422     case 8: //'EC'
423         Display.Digits[3] = 14;
424         Display.Digits[2] = 12;
425         Display.Digits[1] = 35;
426         Display.Digits[0] = 35;
427         break;
428     case 9: //'ED'
429         Display.Digits[3] = 14;
430         Display.Digits[2] = 13;
431         Display.Digits[1] = 35;
432         Display.Digits[0] = 35;
433         break;
434     case 10:    //'EE'
```

```
435         Display.Digits[3] = 14;
436         Display.Digits[2] = 14;
437         Display.Digits[1] = 35;
438         Display.Digits[0] = 35;
439         break;
440     case 11:    //'EEA'
441         Display.Digits[3] = 14;
442         Display.Digits[2] = 14;
443         Display.Digits[1] = 10;
444         Display.Digits[0] = 35;
445         break;
446     case 12:    //'EEB'
447         Display.Digits[3] = 14;
448         Display.Digits[2] = 14;
449         Display.Digits[1] = 11;
450         Display.Digits[0] = 35;
451         break;
452     case 13:    //'EEC'
453         Display.Digits[3] = 14;
454         Display.Digits[2] = 14;
455         Display.Digits[1] = 12;
456         Display.Digits[0] = 35;
457         break;
458     case 14:    //'EED'
459         Display.Digits[3] = 14;
460         Display.Digits[2] = 14;
461         Display.Digits[1] = 13;
462         Display.Digits[0] = 35;
463         break;
464     case 20: // 0uFL
465         Display.Digits[0] = 31;
466         Display.Digits[1] = 15;
467         Display.Digits[2] = 30;
468         Display.Digits[3] = 0;
469         break;
470     case 21: // InuD
471         Display.Digits[0] = 33;
472         Display.Digits[1] = 30;
473         Display.Digits[2] = 32;
474         Display.Digits[3] = 1;
475 
476         break;
477     }
478 }
479 
480 void displayID_SSD(){
481     Display.Digits[0]= 7;
482     Display.Digits[1]= 2;
483     Display.Digits[2]= 7;
484     Display.Digits[3]= 1;
485 
486 
487 }
488 
489 void overflow_SSD(){
490 
491     displaychar_SSD(20);
492 
493     Display.Oflw = 0;
494     calculation.current_process=0;
495     calculation.x=0;
496     calculation.y=0;
```

```
497     calculation.result=0;
498 
499 }
500 
501 void invalid_SSD(void){
502 
503     displaychar_SSD(21);
504 
505     Display.Inv = 0;
506     calculation.current_process=0;
507     calculation.x=0;
508     calculation.y=0;
509     calculation.result=0;
510 
511 }
512 
513 void utility_SSD(float var){
514 
515     int number = (int)var;
516 
517     float i = 0.0;
518 
519     if((number < 0) & (number >= -999)){
520         Display.sign = 1;
521         i = -1.0;
522         Display.dot = 0;
523         if(number >= -99){
524             i = -10.0;
525             Display.dot = 1;
526             if(number >= -9){
527                 i = -100.0;
528                 Display.dot = 2;
529             }
530         }
531     }
532     else if((number >= 0) & (number <= 9999)){
533         Display.sign = 0;
534         i = 1.0;
535         Display.dot = 0;
536         if(number <= 999){
537             i = 10.0;
538             Display.dot = 1;
539             if(number <= 99){
540                 i = 100.0;
541                 Display.dot = 2;
542                 if(number <= 9){
543                     i = 1000.0;
544                     Display.dot = 3;
545                 }
546             }
547         }
548     }
549 
550 
551     number = (int)(var * i);
552 
553     int temp = number / 10;
554     Display.Digits[0] = (uint8_t)(number - (temp*10));
555 
556     temp = number / 100;
557     Display.Digits[1] = (uint8_t)((number - (temp * 100)) / 10);
558 
```

```
559     temp = number / 1000;
560     Display.Digits[2] = (uint8_t)((number - (temp * 1000)) / 100);
561
562     temp = number / 10000;
563     Display.Digits[3] = (uint8_t)((number - (temp * 10000)) / 1000);
564
565     // negative sign
566     if (Display.sign) Display.Digits[3] = 34;
567
568 }
569
570
```

```c
/*
 * calc.h
 *
 *  Created on: Dec 20, 2020
 *      Author: Mehmet Akif/171024027
 */

#ifndef CALC_H_
#define CALC_H_

#include "display.h"
#include "main.h"
/*
 * enum for func
 */
typedef enum{
    none,
    Addition,
    Substraction,
    Multiplacation,
    Division,
    E,
    Log,
    Ln,
    Sqrt,
    Pow2,
    EE,
    Sin,
    Cos,
    Tan,
    Cot
}calculator;

/*
 * the variables struct keep the
 * data which are number1 number2
 * and process number
 */

typedef struct{
    float x;
    float y;
    float result;
    uint8_t current_process;
}variables;

variables calculation;

typedef void (*funcptr)(void);

typedef struct{
    calculator processnumber;
    funcptr    func;
}process;



void calculate();
void nonefunc();
void AdditonFunc();
void SubstractionFunc();
void MultiplacationFunc();
```

```
63 void DivisionFunc();
64 void Efunc();
65 void logFunc();
66 void LnFunc();
67 void SqrtFunc();
68 void Pow2Func();
69 void EEFunc();
70 void SinFunc();
71 void CosFunc();
72 void TanFunc();
73 void CotFunc();
74
75
76 #endif /* CALC_H_ */
77
```

```c
/*
 * calc.c
 *
 *  Created on: Dec 20, 2020
 *      Author: Mehmet Akif/171024027
 */
#include "calc.h"

#define PI 3.141
/*
 * Process has processnumber and
 * functions
 */
process funcArr[] = {
        {none,nonefunc},
        {Addition, AdditonFunc},
        {Substraction, SubstractionFunc},
        {Multiplacation, MultiplacationFunc},
        {Division, DivisionFunc},
        {E, Efunc},
        {Log, logFunc},
        {Ln, LnFunc},
        {Sqrt, SqrtFunc},
        {Pow2, Pow2Func},
        {EE, EEFunc},
        {Sin, SinFunc},
        {Cos, CosFunc},
        {Tan, TanFunc},
        {Cot, CotFunc},
};

/*
 * this func show that which func we have to go
 * according to enum
 */
void calculate(){
    funcArr[calculation.current_process].func();
}

/*none func for enum 0*/
void nonefunc(){
}

void AdditonFunc(){

    float c= calculation.x + calculation.y;
    if(c>9999){
        Display.Oflw=1;
    }
    else calculation.result=c;

}

void SubstractionFunc(){
        float c = calculation.x-calculation.y;

        if(c<-999){
            Display.Oflw=1;
        }
        else calculation.result=c;
}

```

```
63 void MultiplacationFunc(){
64         float c = calculation.x * calculation.y;
65         if((c>9999) | (c<-999)){
66             Display.Oflw=1;
67         }
68         else calculation.result=c;
69 
70 }
71 void DivisionFunc(){
72 
73     if(calculation.y==0){
74         Display.Inv=1;
75     }
76     else  calculation.result = calculation.x/calculation.y;
77 
78 }
79 /*empty func for enum 5*/
80 void Efunc(){
81 }
82 void logFunc(){
83     if(calculation.x==0){
84         Display.Inv=1;
85     }
86     else{
87         calculation.result = log10(calculation.x);
88     }
89 }
90 void LnFunc(){
91     if(calculation.x==0){
92         Display.Inv=1;
93     }
94     else{
95         calculation.result = log(calculation.x);
96     }
97 }
98 void SqrtFunc(){
99     if(calculation.x<0){
100         Display.Inv=1;
101     }
102     else{
103         calculation.result = sqrt(calculation.x);
104     }
105 
106 }
107 /* x^2*/
108 void Pow2Func(){
109     calculation.result = pow(calculation.x,2);
110 }
111 /*empty func for enum 10*/
112 void EEFunc(){
113 }
114 void SinFunc(){
115     float c=(calculation.x * PI) / 180;
116     calculation.result = sin(c);
117 }
118 void CosFunc(){
119     float c=(calculation.x * PI) / 180;
120     calculation.result = cos(c);
121 }
122 void TanFunc(){
123 
124     float c=(calculation.x * PI) / 180;
```

```
125     if(cos(c)==0){
126         Display.Inv=1;
127     }
128     else calculation.result = (sin(c)/cos(c));
129 
130 }
131 void CotFunc(){
132     float c=(calculation.x * PI) / 180;
133     if(sin(c)==0){
134         Display.Inv=1;
135     }
136     else calculation.result = (cos(c)/sin(c));
137 }
138 
139 
140 
141 
142 
```

```
1 /*
2  * bsp.h
3  *
4  *  Created on: 22 Ara 2020
5  *      Author: Mehmet Akif/171024027
6  */
7
8 #ifndef BSP_H_
9 #define BSP_H_
10
11 #include "keypad.h"
12
13 void BSP_init();
14 void init_timer1();
15
16 void delay_ms(volatile unsigned int);
17
18 #endif /* BSP_H_ */
19
```

```c
/*
 * bsp.c
 *
 *  Created on: 22 Ara 2020
 *      Author: Mehmet Akif/171024027
 */
#include "bsp.h"

void BSP_init(){
    __disable_irq();
    keypad_init();
    init_SSD();
    SystemCoreClockUpdate();    //contains the system frequency
    init_timer1();
    displayID_SSD();
        Display.Oflw = 0;
        Display.Inv = 0;
        calculation.current_process=0;
        calculation.x=0;
        calculation.y=0;
        calculation.result=0;
    __enable_irq();
}

void init_timer1(){

    RCC->APBENR2 |= (1U<< 11);// enable time1 module clock

    TIM1->CR1=0;// zero out the control register just in case
    TIM1->CR1 |= (1<<7);    // ARPE
    TIM1->CNT=0;// zero out counter

    /*0.1 ms interrupt   */

    TIM1->PSC=99;
    TIM1->ARR=16;

    TIM1->DIER |= (1 << 0);// update interrupt enable
    TIM1->CR1 |= (1 << 0);//    tım1 enable

    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn,3);
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);

}

void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
{
    TIM1->SR &= ~(1U<<0); //clear update status register

    display_SSD();
}

void delay_ms(volatile unsigned int s){

    for(int i=s; i>0; i--){
     SysTick_Config(SystemCoreClock / 1000); // 16 MHz / 1000 ile 1 ms elde edildi.
     }
}


```

```c
/*
 * main.h
 *
 *  Created on: Dec 19, 2020
 *      Author: Mehmet Akif/171024027
 */

#ifndef MAIN_H_
#define MAIN_H_

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


#endif /* MAIN_H_ */

```

```c
/*
 * main.c
 *
 * Author: Mehmet Akif/171024027
 *
 * description:
 */

#include "main.h"
#include "bsp.h"


int main(){

    BSP_init();


    return 0;
}

```