GEBZE TECHNICAL UNIVERSITY

ENGINEERING FACULTY

ELECTRONICS ENGINEERING

ELEC 334

MICROPROCESSORS

PROJECT #2

Name - Surname	Yağmur DERYA
Student ID	171024011

Introduction

In this project, the goal is to create a fully operational scientific calculator, which has a keypad connected to enter the numbers and functions and a 4 -digit SSD to display numbers, in C.

Project

Parts list w/prices:

• Breadboard	7.27	TL
• Nucleo-G031K8	≈ 100	TL
• Jumper (x40)	3.06	TL
• 4-Digit Seven Segment Display	6.12	TL
• Resistor (470Ω x 4)	4x0.02	TL
• Micro USB Cable	20	TL

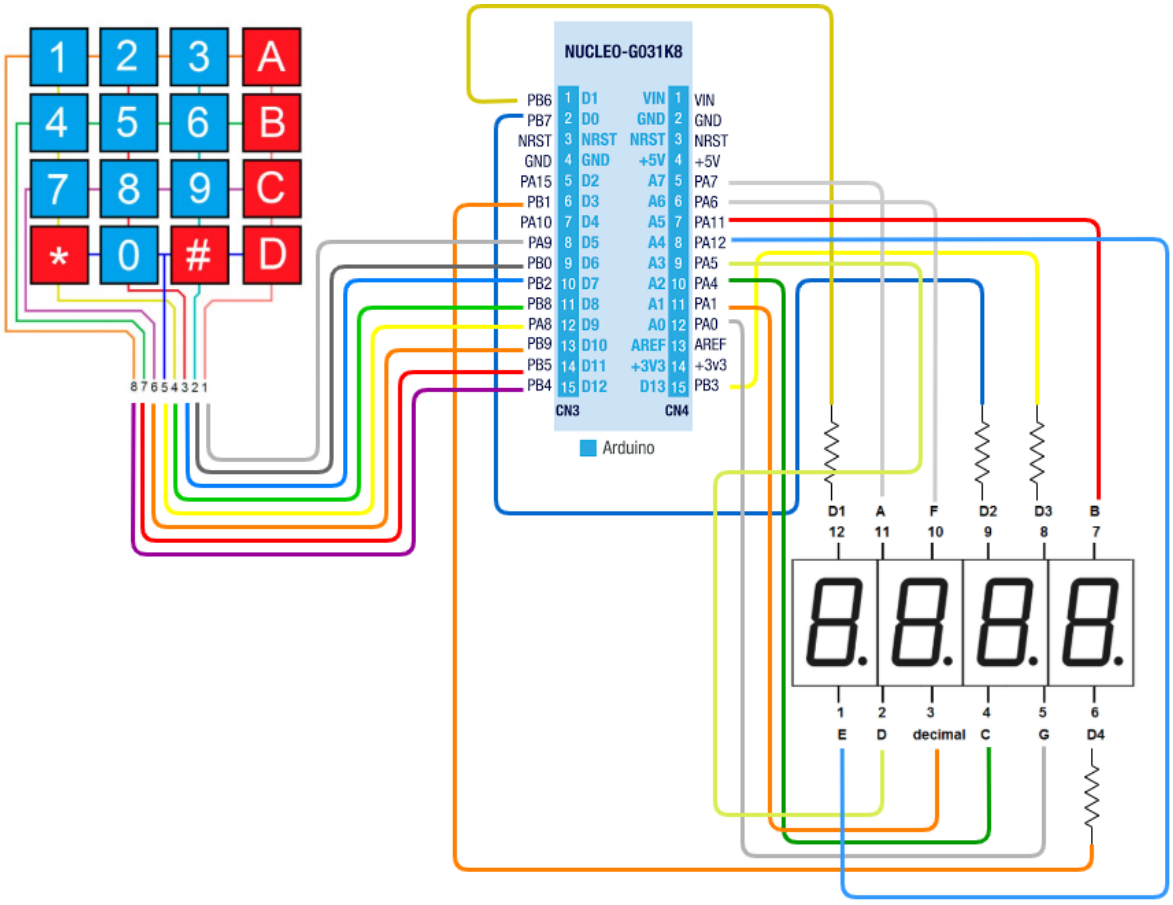

Figure 1. Connection diagram of the project.

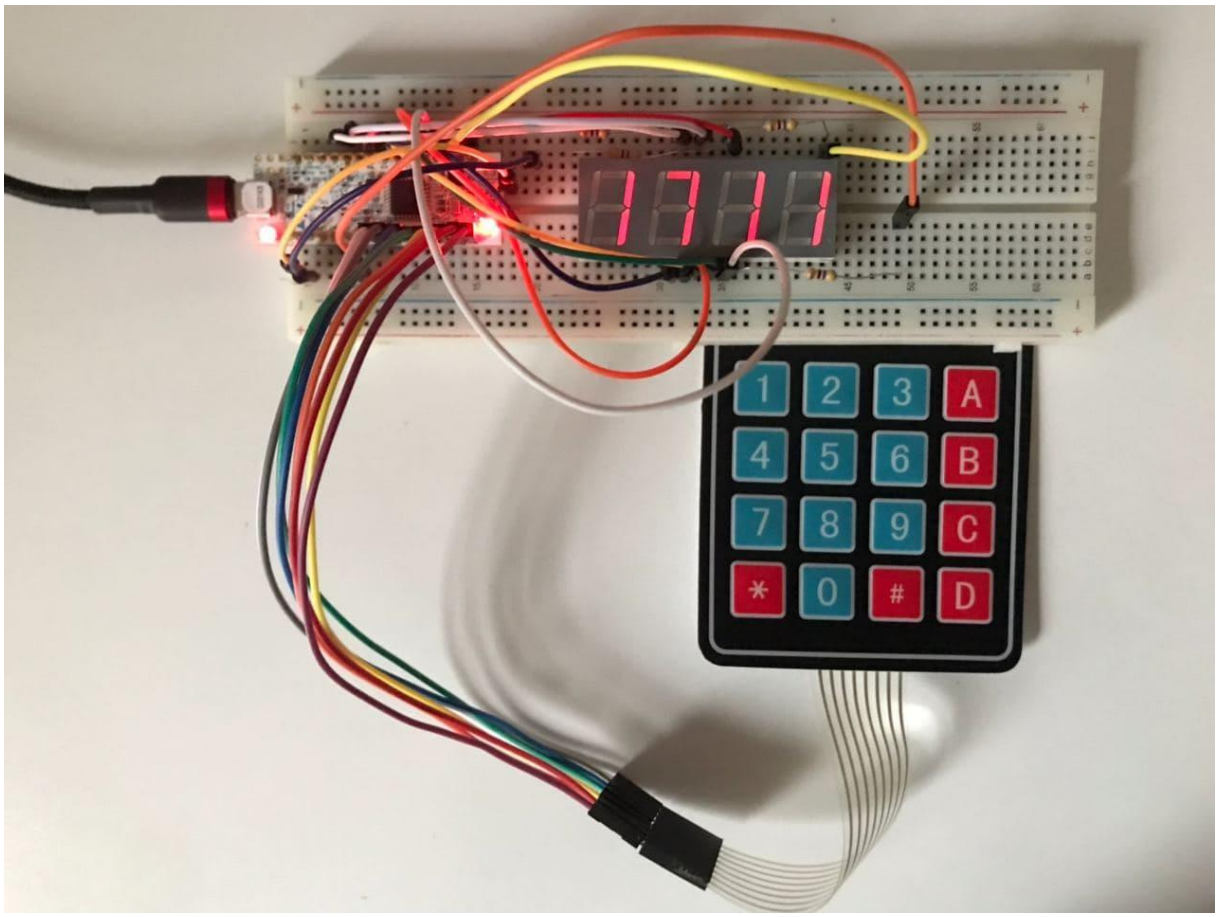

Figure 2. Project setup.

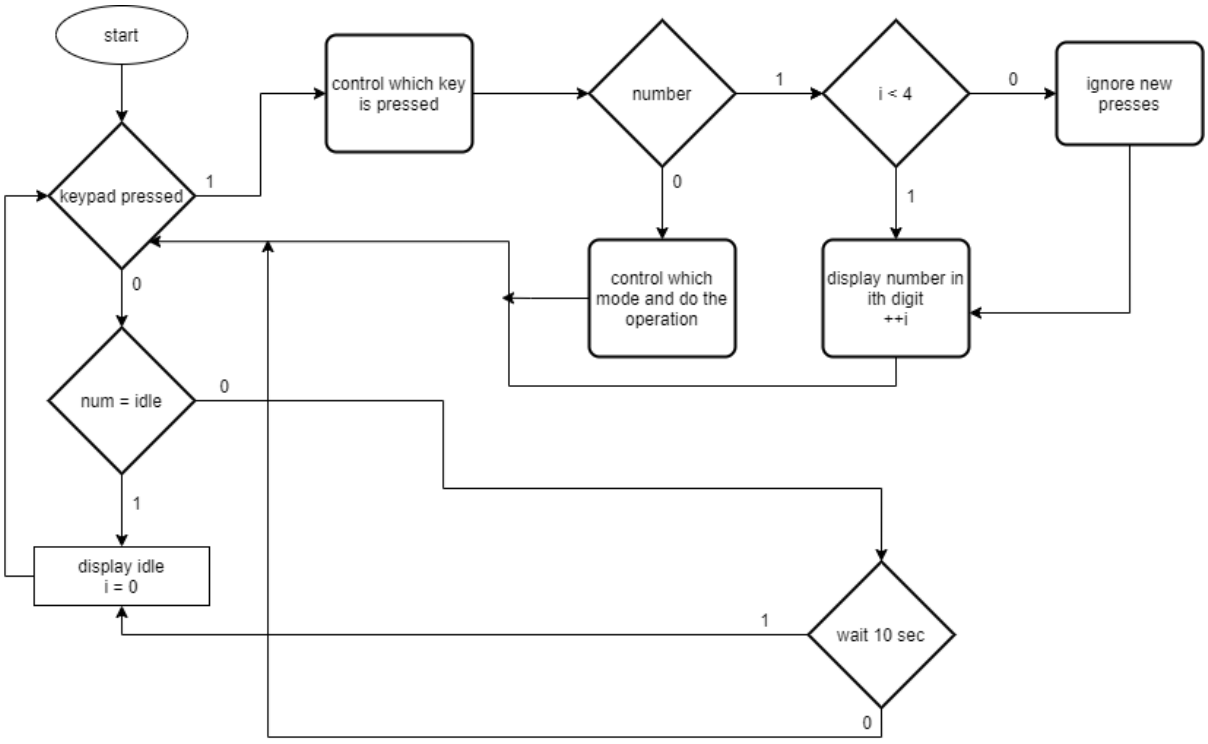

Figure 3. Basic flowchart.

Tasks

1. Connect SSD and display numbers on 4-digit. ✓
2. Display school ID as IDLE. ✓
3. Connect Keypad and get inputs correctly. ✓
4. Check modes. ✓
5. Shift the numbers. ✓
6. Ignore new key presses if digits are full. ✓
7. Go back to IDLE state if no button is pressed for 10 seconds. ✓
8. Calculate with integer numbers. ✓
9. Display double number on SSD. ✓
10. Calculate with double numbers. ✓
11. Check if scientific modes work true with dump variables. ✓
12. Display integer number if fractal = 0 else display decimal. ✓
13. Control if the last digit of fractal = 0. If it is, do not display that digit. ✓
14. Write equations for scientific calculations. ✓
15. Keep the last value if directly a function is invoked. ✓

Methodology

I defined pin numbers in my header file so I don't need to control pin numbers every time and if I want to change my pin, I don't need to change number in every related line, just change in my header one time.

After connecting SSD and Keypad and get inputs and outputs correctly, I added flags to understand which mode is active. By these set and reset these flags the code goes to the related condition and do the calculation etc.

I used timer (TIM1) for my infinite loop. My external interrupts are in highest priority (0) and TIM1 is in second highest priority (1). If any interrupt happens, the code lefts TIM1 handler and goes related handler. All calculations and set-reset flags happen in EXTI handler part. Basically, I set or reset flags in every related function to understand modes and do the correct operation.

To write numbers in right digits and count how many digits the number has, numbers should be formed as arrays. I used string as array to do these operations. I defined `num[]` array and get the numbers as string. Then I define an index flag to get the true digit of the numbers and then change the type string to float by "`atof()`" function in "`stdlib.h`". To write a number as string in my `num[]` array, I used "`gcvy()`" function. I used "`math.h`" library to calculate in scientific mode. Since there is no cot function in this library and cot = 1/tan, I used tan function to calculate cot. Also, I added conditions for invalid numbers.

I used float numbers for all operations, but I display the number as integer if it does not have decimal part, by checking fractal value.

First, I used Systick Handler to create a delay function like I did in Lab0x4 before, but it does not work as expected in my code. I tried to solve the problem by debug the project and control what is wrong but could not fix it, so I wrote a delay function with three for loops. It has three loops because for debouncing the code needs more than one 100000 loops and when I increased the number, I got infinite loop error although I defined the variable as long/long long it. So I added three for loops to solve this problem and do not call delay function repeatedly.

Conclusion

Project completed successfully. All modes work as they wanted to be. In this project I learned new functions about strings, the importance of using flags and converting string to integer or float. Delay with for loop changes by codes length so since it is unstable, using for loop or while loop is not a good way for create delay. I understand interrupts priorities and handlers better in this project.

References

[1] RM0444 Reference manuel
[2] https://github.com/fcayci/stm32g0
[3] https://pdf.direnc.net/upload/14mm-4-lu-ortak-anot-display-datasheet.pdf
[4] https://www.tutorialspoint.com/c_standard_library/c_function_strcmp.htm
[5] https://www.tutorialspoint.com/c_standard_library/c_function_atof.htm
[6] https://www.geeksforgeeks.org/gcvt-convert-float-value-string-c/
[7] https://www.tutorialspoint.com/c_standard_library/c_function_modf.htm
[8] https://www.tutorialspoint.com/c_standard_library/c_function_strcpy.htm

Appendix A – Used Library & Function List

```
<string.h>
int strcmp(cons char *str1, const char *str2);
char *strcpy(char *dest, const char *source);
```

```
<math.h>
double modf(double x, double *integer);

double log10(double x);
double log(double x);
double sqrt(double x);
double pow(double x, double y);
double sin(double x);
double cos(double x);
double tan(double x);
```

```
<stdlib.h>
double atof(const char *str);
```

```
<stdio.h>
gcvt(float value, int ndigits, char *buf);
```

Appendix B – Code Listing

```
/*
 * main.c
 *
 *           Yagmur Derya
 *           171024011
 *
 */

#include "project2.h"

int main(void) {
      init_system();

    while(1){
    }
    return 0;
}
```

```
/*
 * project2.h
 *
 *  Yagmur Derya
 *  171024011
 *
 */

#ifndef PROJECT2_H_
#define PROJECT2_H_

/* defines for SSD */
#define D1      6  // PB
#define D2      7  // PB
#define D3      3  // PB
#define D4      1  // PB
#define A       7  // PA
#define B       11 // PA
#define c       4  // PA
#define D       5  // PA
#define E       12 // PA
#define F       6  // PA
#define G       0  // PA
#define decimal 1  // PA

/* defines for Keypad */
#define R1      4  // PB
#define R2      5  // PB
#define R3      9  // PB
#define R4      8  // PA
#define C1      8  // PB
#define C2      2  // PB
#define C3      0  // PB
#define C4      9  // PA

#include "stm32g0xx.h"

void init_system(void);

void EXTI0_1_IRQHandler(void);
```

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

void TIM1_BRK_UP_TRG_COM_IRQHandler(void);
void init_timer1(void);

void delay(void);

void clear_array(char*);

/* setup for both SSD and Keypad */
void init_pins(void);

/* for SSD */
void set_SSD(int);
void clear_SSD(void);
void set_digit(int);
void clear_digit(int);
void clear_all_digits(void);
void toggle_digit(int);

int  digit_num(int);
void overflow(void);
void invalid(void);
void display_negative(int, int);
void display_positive(int, int);
void display(double);

/* for Keypad - PB4 PB5 PB9 PA8 PB8 PB2 PB0 PA9 */
void set_rows(void);
void clear_rows(void);

#endif /* PROJECT2_H_ */
```

```
/*
 * project2.c
 *
 *  Yagmur Derya
 *  171024011
 *
 */

#include "project2.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* all of them global so they are 0 */
int wait;  // wait 10 secs before go back to idle
int ind;   // digit index = 0
char num[]  = "1711";
char arr[]  = " ";
double result;
int inv;

int add;       // A
int sub;       // B
int mul;       // C
```

```
int divi;       // D
int smode;      // E
int smode_c;    // counter for scientific mode
int lg;         // EA
int ln;         // EB
int sroot;      // EC
int square;     // ED
int trigo;      // EE
int sinus;      // EEA
int cosinus;    // EEB
int tangent;    // EEC
int cot;        // EED
int pi;         // EEE (replace the number with 3.141)
```

```
void init_system(void) {
	init_pins();
	SysTick_Config(SystemCoreClock / 1000); // 1 ms interrupt interval = 16MHz/1ms = 16 000 = SystemCoreClock/1000
	init_timer1();
}
```

```
void EXTI0_1_IRQHandler(void) {		// PB0 : C3
	if (!strcmp(num, "1711")) {		// if num = 1711, returns 0
		clear_array(num);			// clear IDLE state
	}

	clear_rows();

	GPIOB->ODR ^= (1U << R1);		// PB4 = 1 - R1
	if(((GPIOB->IDR >> C3) & 1) && ind < 4) {
		// 3
		if(ind == 0) {
			clear_array(num);
		}
		num[ind] = '3';
		ind ++;
		wait = 0;
		delay();
	} GPIOB->ODR ^= (1U << R1);		// PB4 = 0

	GPIOB->ODR ^= (1U << R2);		// PB5 = 1 - R2
	if(((GPIOB->IDR >> C3) & 1) && ind < 4) {
		// 6
		if(ind == 0) {
			clear_array(num);
		}
		num[ind] = '6';
		ind ++;
		wait = 0;
		delay();
	} GPIOB->ODR ^= (1U << R2);		// PB5 = 0

	GPIOB->ODR ^= (1U << R3);		// PB9 = 1 - R3
	if(((GPIOB->IDR >> C3) & 1) && ind < 4) {
		// 9
		if(ind == 0) {
			clear_array(num);
		}
		num[ind] = '9';
		ind ++;
		wait = 0;
```

```
        delay();
    } GPIOB->ODR ^= (1U << R3);         // PB9 = 0

    GPIOA->ODR ^= (1U << R4);           // PA8 = 1 - R4
    if((GPIOB->IDR >> C3) & 1 ){
        // # : =
        smode_c = 0;
        if (add) {
            add = 0;
            if (pi) {
                result = 3.141;
                gcvt(result, 4, num);
                pi = 0;
            }
            result = atof(arr) + atof(num);
            gcvt(result, 4, num);
        } else if (sub) {
            sub = 0;
            if (pi) {
                result = 3.141;
                gcvt(result, 4, num);
                pi = 0;
            }
            result = atof(arr) - atof(num);
            gcvt(result, 4, num);
        } else if (mul) {
            mul = 0;
            if (pi) {
                result = 3.141;
                gcvt(result, 4, num);
                pi = 0;
            }
            result = atof(arr) * atof(num);
            gcvt(result, 4, num);
        } else if (divi) {
            divi = 0;
            if (pi) {
                result = 3.141;
                gcvt(result, 4, num);
                pi = 0;
            }
            if (atof(num) == 0.000) {
                inv = 1;
            } else {
                result = atof(arr) / atof(num);
                gcvt(result, 4, num);
            }
        } else if (smode) {
            smode = 0;
            if (pi) {
                result = 3.141;
                gcvt(result, 4, num);
                pi = 0;
            }
            if (lg) {
                lg = 0;
                if (atof(num) <= 0.000){
                    inv = 1;
                } else {
```

```
                        result = log10(atof(num));
                        gcvt(result, 4, num);
                }
        } else if (ln) {
                ln = 0;
                if (pi) {
                        result = 3.141;
                        gcvt(result, 4, num);
                        pi = 0;
                }
                if (atof(num) <= 0.000) {
                        inv = 1;
                } else {
                        result = log(atof(num));
                        gcvt(result, 4, num);
                }
        } else if (sroot) {
                sroot = 0;
                if (pi) {
                        result = 3.141;
                        gcvt(result, 4, num);
                        pi = 0;
                }
                if(atof(num) < 0){
                        inv = 1;
                } else {
                        result = sqrt(atof(num));
                        gcvt(result, 4, num);
                }
        } else if (square) {
                square = 0;
                if (pi) {
                        result = 3.141;
                        gcvt(result, 4, num);
                        pi = 0;
                }
                        result = pow(atof(num), 2);
                        gcvt(result, 4, num);
        }
} else if (trigo) {
        trigo = 0;
        if (sinus) {
                /* radian */
                sinus = 0;
                if (pi) {
                        result = 3.141;
                        gcvt(result, 4, num);
                        pi = 0;
                }
                result = sin(atof(num));
                gcvt(result, 4, num);
        } else if (cosinus) {
                /* radian */
                cosinus = 0;
                if (pi) {
                        result = 3.141;
                        gcvt(result, 4, num);
                        pi = 0;
                }
```

```
                        result = cos(atof(num));
                        gcvt(result, 4, num);
                    } else if (tangent) {
                        /* radian */
                        tangent = 0;
                        if (pi) {
                            result = 3.141;
                            gcvt(result, 4, num);
                            pi = 0;
                        }
                        if (cos(atof(num)) == 0) {
                            /* tan = (sin / cos) */
                            inv = 1;
                        } else {
                            result = tan(atof(num));
                            gcvt(result, 4, num);
                        }
                    } else if (cot) {
                        /* radian */
                        cot = 0;
                        if (pi) {
                            result = 3.141;
                            gcvt(result, 4, num);
                            pi = 0;
                        }
                        if (sin(atof(num)) == 0) {
                            /* cot = (cos / sin) = 1 / tan */
                            inv = 1;
                        } else {
                            result = 1 / tan(atof(num));
                            gcvt(result, 4, num);
                            pi = 0;
                        }
                    }
                } else if (pi) {
                    result = 3.141;
                    gcvt(result, 4, num);
                    pi = 0;
                }
                wait = 0;
                delay();
                ind = 0;
        } GPIOA->ODR ^= (1U << R4);         // PA8 = 0

        EXTI->RPR1 |= (1U << C3);           // clear pending
        set_rows();
}
```

```
void EXTI2_3_IRQHandler(void) {         // PB2 : C2
        if (!strcmp(num, "1711")) {         // if num = 1711, returns 0
                clear_array(num);               // clear IDLE state
        }

        clear_rows();

        GPIOB->ODR ^= (1U << R1);           // PB4 = 1 - R1
        if(((GPIOB->IDR >> C2) & 1) && ind < 4) {
                // 2
                if(ind == 0) {
                        clear_array(num);
```

```
		}
		num[ind] = '2';
		ind ++;
		wait = 0;
		delay();
	} GPIOB->ODR ^= (1U << R1);		// PB4 = 0

	GPIOB->ODR ^= (1U << R2);		// PB5 = 1 - R2
	if(((GPIOB->IDR >> C2) & 1) && ind < 4) {
		// 5
		if(ind == 0) {
			clear_array(num);
		}
		num[ind] = '5';
		ind ++;
		wait = 0;
		delay();
	} GPIOB->ODR ^= (1U << R2);		// PB5 = 0

	GPIOB->ODR ^= (1U << R3);		// PB9 = 1 - R3
	if(((GPIOB->IDR >> C2) & 1) && ind < 4) {
		// 8
		if(ind == 0) {
			clear_array(num);
		}
		num[ind] = '8';
		ind ++;
		wait = 0;
		delay();
	} GPIOB->ODR ^= (1U << R3);		// PB9 = 0

	GPIOA->ODR ^= (1U << R4);		// PA8 = 1 - R4
	if (((GPIOB->IDR >> C2) & 1) && ind < 4) {
		// 0
		if(ind == 0) {
			clear_array(num);
		}
		num[ind] = '0';
		ind ++;
		wait = 0;
		delay();
	} GPIOA->ODR ^= (1U << R4);		// PA8 = 0

	EXTI->RPR1 |= (1U << C2);		// clear pending
	set_rows();
}
```

```
void EXTI4_15_IRQHandler(void) {
	if (!strcmp(num, "1711")) {		// if num = 1711, returns 0
		clear_array(num);			// clear IDLE state
	}

	/* PB8 : C1 */
	if ((GPIOB->IDR >> C1) & 1) {
		clear_rows();

		GPIOB->ODR ^= (1U << R1);	// PB4 = 1 - R1
		if (((GPIOB->IDR >> C1) & 1) && ind < 4) {
			// 1
			if(ind == 0) {
```

```
                        clear_array(num);
                }
                num[ind] = '1';
                ind ++;
                wait = 0;
                delay();
        } GPIOB->ODR ^= (1U << R1);     // PB4 = 0

        GPIOB->ODR ^= (1U << R2);       // PB5 = 1 - R2
        if (((GPIOB->IDR >> C1) & 1) && ind < 4) {
                // 4
                if(ind == 0) {
                        clear_array(num);
                }
                num[ind] = '4';
                ind++;
                wait = 0;
                delay();
        } GPIOB->ODR ^= (1U << R2);     // PB5 = 0

        GPIOB->ODR ^= (1U << R3);       // PB9 = 1 - R3
        if (((GPIOB->IDR >> C1) & 1) && ind < 4) {
                // 7
                if(ind == 0) {
                        clear_array(num);
                }
                num[ind] = '7';
                ind ++;
                wait = 0;
                delay();
        } GPIOB->ODR ^= (1U << R3);     // PB9 = 0

        GPIOA->ODR ^= (1U << R4);       // PA8 = 1 - R4
        if ((GPIOB->IDR >> C1) & 1) {
                // *
                // scientific mode
                smode = 0;
                trigo = 0;
                pi = 0;
                if (smode_c == 0) {
                        smode = 1;
                        smode_c ++ ;
                } else if (smode_c == 1) {
                        trigo = 1;
                        smode_c ++;
                } else if (smode_c == 2) {
                        smode_c = 0;
                        pi = 1;
                }
                wait = 0;
                ind = 0;
                clear_array(num);
                delay();
        } GPIOA->ODR ^= (1U << R4);     // PA8 = 0
}
/* PA9 : C4 */
else if ((GPIOA->IDR >> C4) & 1){
        clear_rows();
```

```
            GPIOB->ODR ^= (1U << R1);       // PB4 = 1 - R1
            if ((GPIOA->IDR >> C4) & 1) {
                  // A
                  if (smode) {
                        lg = 1;                 // EA  : log
                  } else if (trigo) {
                        sinus = 1;              // EEA : sin
                  } else {
                        add = 1;                // addition
                  }
            } GPIOB->ODR ^= (1U << R1);     // PB4 = 0

            GPIOB->ODR ^= (1U << R2);       // PB5 = 1 - R2
            if ((GPIOA->IDR >> C4) & 1) {
                  // B
                  if (smode) {
                        ln = 1;                 // EB  : ln
                  } else if (trigo) {
                        cosinus = 1;            // EEB : cos
                  } else {
                        sub = 1;                // subtraction
                  }
            } GPIOB->ODR ^= (1U << R2);     // PB5 = 0

            GPIOB->ODR ^= (1U << R3);       // PB9 = 1 - R3
            if ((GPIOA->IDR >> C4) & 1) {
                  // C
                  if (smode) {
                        sroot = 1;              // EC  : sqrt
                  } else if (trigo) {
                        tangent = 1;            // EEC : tan
                  } else {
                        mul = 1;                // multiplication
                  }
            } GPIOB->ODR ^= (1U << R3);     // PB9 = 0

            GPIOA->ODR ^= (1U << R4);       // PA8 = 1 - R4
            if ((GPIOA->IDR >> C4) & 1) {
                  // D
                  if (smode) {
                        square = 1;             // ED  : x^2
                  } else if (trigo) {
                        cot = 1;                // EED : cot
                  } else {
                        divi = 1;               // division
                  }
            } GPIOA->ODR ^= (1U << R4);     // PA8 = 0
            wait = 0;
            ind = 0;
            strcpy(arr, num);
            clear_array(num);
            delay();
      }

      EXTI->RPR1 |= (1U << C4);           // clear pending
      set_rows();
}
```

```
void init_timer1(void) {
      RCC->APBENR2 |= (1U << 11); // TIM1  clock enable
```

```
	TIM1->CR1 = 0;              // control register reset
	TIM1->CR1 |= (1 << 7);      // ARPE
	TIM1->CNT = 0;              // zero out counter

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

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

	NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 1);
	NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
}
```

```
void TIM1_BRK_UP_TRG_COM_IRQHandler(void) {
	++ wait;
	display(atof(num));
	if (result > 9999.0 || result < -999.0) {
		for (int i = 0; i < 10000; ++i) {
			overflow();
		}
		wait = 0;
		ind = 0;
		result = 0;
		inv = 0;
		strcpy(num, "1711");  // num = 1711
	}
	if (inv) {
		for (int i = 0; i < 10000; ++i) {
			invalid();
		}
		wait = 0;
		ind = 0;
		result = 0;
		inv = 0;
		strcpy(num, "1711");  // num = 1711
	}

	if (wait == 50000) {     // almost 10 secs wait
		wait = 0;
		ind = 0;
		result = 0;
		inv = 0;
		strcpy(num, "1711");
	}
	TIM1->SR &= ~(1U << 0); // clear update status
}
```

```
void delay(void) {
	for(int s = 0; s < 100000; ++s);
	for(int s = 0; s < 100000; ++s);
	for(int s = 0; s < 80000; ++s);
}
```

```
void clear_array(char* a){
	for (int i = 0; i < 4; ++i) {
		a[i] = '\0';                  // clear by making them NULL
	}
}
```

```
void init_pins(void) {
	RCC->IOPENR |= (3U << 0);        // enable GPIOA and GPIOB (0011 = 3)
```

```
    /* setup for SSD inputs on PB1 PB3 PB7 PB6 */
    GPIOB->MODER &= ~(3U << 2*D1);   // PB6  - D1
    GPIOB->MODER |=  (1U << 2*D1);

    GPIOB->MODER &= ~(3U << 2*D2);   // PB7  - D2
    GPIOB->MODER |=  (1U << 2*D2);

    GPIOB->MODER &= ~(3U << 2*D3);   // PB3  - D3
    GPIOB->MODER |=  (1U << 2*D3);

    GPIOB->MODER &= ~(3U << 2*D4);   // PB1  - D4
    GPIOB->MODER |=  (1U << 2*D4);

    /* setup for SSD outputs on PA0 PA1 PA4 PA5 PA6 PA7 PA11 PA12 */
    GPIOA->MODER &= ~(3U << 2*decimal);   // reset PA4 - decimal point
    GPIOA->MODER |=  (1U << 2*decimal);   // set for output mode

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

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

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

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

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

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

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

    /* setup for Keypad inputs on PB0 PB2 PB8 PA9 */
    GPIOB->MODER &= ~(3U << 2*C1);   // PB8  - C1
    GPIOB->PUPDR |=  (2U << 2*C1);   // Pull-down mode.

    GPIOB->MODER &= ~(3U << 2*C2);   // PB2  - C2
    GPIOB->PUPDR |=  (2U << 2*C2);

    GPIOB->MODER &= ~(3U << C3);     // PB0  - C3
    GPIOB->PUPDR |=  (2U << C3);

    GPIOA->MODER &= ~(3U << 2*C4);   // PA9  - C4
    GPIOA->PUPDR |=  (2U << 2*C4);

    /* Setup interrupts */
    EXTI->EXTICR[2] |= (1U << 0);   // B8
    EXTI->EXTICR[0] |= (1U << 8*2); // B2
    EXTI->EXTICR[0] |= (1U << 0);   // B0
    EXTI->EXTICR[2] |= (0U << 8*1); // A9

    /* rising edge */
```

```
	EXTI->RTSR1	|= (1U << 8);	// B8
	EXTI->RTSR1	|= (1U << 2);	// B2
	EXTI->RTSR1	|= (1U << 0);	// B0
	EXTI->RTSR1	|= (1U << 9);	// A9

	/* Mask */
	EXTI->IMR1	|= (1U << 8);	// B8
	EXTI->IMR1	|= (1U << 2);	// B2
	EXTI->IMR1	|= (1U << 0);	// B0
	EXTI->IMR1	|= (1U << 9);	// A9

	/* NVIC all in high 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 for Keypad outputs on PB4 PB5 PB9 PA8 */
	GPIOB->MODER &= ~(3U << 2*R1);	// PB4  - R1
	GPIOB->MODER |=  (1U << 2*R1);

	GPIOB->MODER &= ~(3U << 2*R2);	// PB5  - R2
	GPIOB->MODER |=  (1U << 2*R2);

	GPIOB->MODER &= ~(3U << 2*R3);	// PB9  - R3
	GPIOB->MODER |=  (1U << 2*R3);

	GPIOA->MODER &= ~(3U << 2*R4);	// PA8  - R4
	GPIOA->MODER |=  (1U << 2*R4);

	/* set all raws */
	GPIOB->ODR   |= (1U << R1);	// PB4
	GPIOB->ODR   |= (1U << R2);	// PB5
	GPIOB->ODR   |= (1U << R3);	// PB9
	GPIOA->ODR   |= (1U << R4);	// PA8

	clear_SSD();
}
```

```
/* for SSD */
void set_SSD(int n) {
	clear_SSD(); // all pins = 1

	switch(n){
	case 0: // abcdef
		GPIOA->ODR &= ~(1U << A);	// turn on A
		GPIOA->ODR &= ~(1U << B);	// turn on B
		GPIOA->ODR &= ~(1U << c);	// turn on C
		GPIOA->ODR &= ~(1U << D);	// turn on D
		GPIOA->ODR &= ~(1U << E);	// turn on E
		GPIOA->ODR &= ~(1U << F);	// turn on F
		break;
	case 1: // bc
		GPIOA->ODR &= ~(1U << B);	// turn on B
		GPIOA->ODR &= ~(1U << c);	// turn on C
		break;
```

```
        case 2: // abdeg
                GPIOA->ODR &= ~(1U << A);         // turn on A
                GPIOA->ODR &= ~(1U << B);         // turn on B
                GPIOA->ODR &= ~(1U << D);         // turn on D
                GPIOA->ODR &= ~(1U << E);         // turn on E
                GPIOA->ODR &= ~(1U << G);         // turn on G
                break;
        case 3: // abcdg
                GPIOA->ODR &= ~(1U << A);         // turn on A
                GPIOA->ODR &= ~(1U << B);         // turn on B
                GPIOA->ODR &= ~(1U << c);         // turn on C
                GPIOA->ODR &= ~(1U << D);         // turn on D
                GPIOA->ODR &= ~(1U << G);         // turn on G
                break;
        case 4: // bcfg
                GPIOA->ODR &= ~(1U << B);         // turn on B
                GPIOA->ODR &= ~(1U << c);         // turn on C
                GPIOA->ODR &= ~(1U << F);         // turn on F
                GPIOA->ODR &= ~(1U << G);         // turn on G
                break;
        case 5: // acdfg
                GPIOA->ODR &= ~(1U << A);         // turn on A
                GPIOA->ODR &= ~(1U << c);         // turn on C
                GPIOA->ODR &= ~(1U << D);         // turn on D
                GPIOA->ODR &= ~(1U << F);         // turn on F
                GPIOA->ODR &= ~(1U << G);         // turn on G
                break;
        case 6: // acdefg
                GPIOA->ODR &= ~(1U << A);         // turn on A
                GPIOA->ODR &= ~(1U << c);         // turn on C
                GPIOA->ODR &= ~(1U << D);         // turn on D
                GPIOA->ODR &= ~(1U << E);         // turn on E
                GPIOA->ODR &= ~(1U << F);         // turn on F
                GPIOA->ODR &= ~(1U << G);         // turn on G
                break;
        case 7: // abc
                GPIOA->ODR &= ~(1U << A);         // turn on A
                GPIOA->ODR &= ~(1U << B);         // turn on B
                GPIOA->ODR &= ~(1U << c);         // turn on C
                break;
        case 8: // abcdefg
                GPIOA->ODR &= ~(1U << A);         // turn on A
                GPIOA->ODR &= ~(1U << B);         // turn on B
                GPIOA->ODR &= ~(1U << c);         // turn on C
                GPIOA->ODR &= ~(1U << D);         // turn on D
                GPIOA->ODR &= ~(1U << E);         // turn on E
                GPIOA->ODR &= ~(1U << F);         // turn on F
                GPIOA->ODR &= ~(1U << G);         // turn on G
                break;
        case 9: // abcdfg
                GPIOA->ODR &= ~(1U << A);         // turn on A
                GPIOA->ODR &= ~(1U << B);         // turn on B
                GPIOA->ODR &= ~(1U << c);         // turn on C
                GPIOA->ODR &= ~(1U << D);         // turn on D
                GPIOA->ODR &= ~(1U << F);         // turn on F
                GPIOA->ODR &= ~(1U << G);         // turn on G
                break;
        case 10: // negative
                GPIOA->ODR &= ~(1U << G);
```

```
            break;
      case 11: // decimal point
            GPIOA->ODR &= ~(1U << decimal);
            break;
      }
}
```

```
void clear_SSD(void) {
      GPIOA->ODR |= (1U << A);              // A             : PA7
      GPIOA->ODR |= (1U << B);              // B             : PA1
      GPIOA->ODR |= (1U << c);              // C             : PA4
      GPIOA->ODR |= (1U << D);              // D             : PA5
      GPIOA->ODR |= (1U << E);              // E             : PA12
      GPIOA->ODR |= (1U << F);              // F             : PA6
      GPIOA->ODR |= (1U << G);              // G             : PA0
      GPIOA->ODR |= (1U << decimal);        // DECIMAL POINT : PA1
}
```

```
void set_digit(int d) {
      if       (d == 4) {
            GPIOB->ODR |= (1U << D4); // D4 : PB1
      } else if(d == 3) {
            GPIOB->ODR |= (1U << D3); // D3 : PB3
      } else if(d == 2) {
            GPIOB->ODR |= (1U << D2); // D2 : PB7
      } else if(d == 1) {
            GPIOB->ODR |= (1U << D1); // D1 : PB6
      }
}
```

```
void clear_digit(int d) {
      if       (d == 4) {
            GPIOB->ODR &= ~(1U << D4); // D4 : PB1
      } else if(d == 3) {
            GPIOB->ODR &= ~(1U << D3); // D3 : PB3
      } else if(d == 2) {
            GPIOB->ODR &= ~(1U << D2); // D2 : PB7
      } else if(d == 1) {
            GPIOB->ODR &= ~(1U << D1); // D1 : PB6
      }
}
```

```
void clear_all_digits(void) {
      for(int i = 1; i <5; ++i) {
            clear_digit(i);            // clears all digits
            ++i;
      }
}
```

```
void toggle_digit(int d) {
      set_digit(d);
      clear_digit(d);
}
```

```
int  digit_num(int n) {     // to count how many digits number has
      int digit = 0;
      if (n == 0){
            return 1;           // 0 has one digit and 0/x gives error.
      }
      while (n != 0) {
            n = n / 10;         // since n is int, this method works for
digit count.
            ++ digit;
      }
            return digit;
```

```
}
void overflow(void) {
	/* 0uFL */
	/* 0 */
	set_SSD(0);
	toggle_digit(1);

	/* u : cde */
	clear_SSD();
	GPIOA->ODR &= ~(1U << c); // turn on C
	GPIOA->ODR &= ~(1U << D); // turn on D
	GPIOA->ODR &= ~(1U << E); // turn on E
	toggle_digit(2);

	/* F : aefg */
	clear_SSD();
	GPIOA->ODR &= ~(1U << A); // turn on A
	GPIOA->ODR &= ~(1U << E); // turn on E
	GPIOA->ODR &= ~(1U << F); // turn on F
	GPIOA->ODR &= ~(1U << G); // turn on G
	toggle_digit(3);

	/* L : def */
	clear_SSD();
	GPIOA->ODR &= ~(1U << D); // turn on D
	GPIOA->ODR &= ~(1U << E); // turn on E
	GPIOA->ODR &= ~(1U << F); // turn on F
	toggle_digit(4);
}
void invalid(void) {
	/* Invd */
	/* I = 1 */
	set_SSD(1);
	toggle_digit(1);

	/* n : ceg */
	clear_SSD();
	GPIOA->ODR &= ~(1U << c); // turn on C
	GPIOA->ODR &= ~(1U << E); // turn on E
	GPIOA->ODR &= ~(1U << G); // turn on G
	toggle_digit(2);

	/* v : cde */
	clear_SSD();
	GPIOA->ODR &= ~(1U << c); // turn on C
	GPIOA->ODR &= ~(1U << D); // turn on D
	GPIOA->ODR &= ~(1U << E); // turn on E
	toggle_digit(3);

	/* d : bcdeg */
	clear_SSD();
	GPIOA->ODR &= ~(1U << B); // turn on B
	GPIOA->ODR &= ~(1U << c); // turn on C
	GPIOA->ODR &= ~(1U << D); // turn on D
	GPIOA->ODR &= ~(1U << E); // turn on E
	GPIOA->ODR &= ~(1U << G); // turn on G
	toggle_digit(4);
}
void display_negative(int n, int f) {
```

```
	n = abs(n);                    // n = |n|
	switch(digit_num(n)) {
	case 1:
		if (((f / 10) % 10) == 0) {
			if (f == 0) {
				set_SSD(n);              // if n = 2    =>
set_SSD(2)
				toggle_digit(4);         //                on D4

				set_SSD(10);
				toggle_digit(3);         // -2
			} else {
				set_SSD(n);              // if n =-2.80  =>
set_SSD(2)
				toggle_digit(3);         //                 on D3

				set_SSD(10);
				toggle_digit(2);         // -2

				set_SSD(11);             // -2.
				toggle_digit(3);

				set_SSD(f / 100);        //              =>
set_SSD(8)
				toggle_digit(4);         // -2.8            on D4
			}

		} else {
			set_SSD(n);              // if n =-2.83  => set_SSD(2)
			toggle_digit(2);         //                 on D2

			set_SSD(10);
			toggle_digit(1);         // -2

			set_SSD(11);             // -2.
			toggle_digit(2);

			set_SSD(f / 100);        //              => set_SSD(8)
			toggle_digit(3);         // -2.8            on D3

			set_SSD((f / 10) % 10);  //              => set_SSD(3)
			toggle_digit(4);         // -2.83           on D4
		}
		break;
	case 2:
		if(f == 0) {
			set_SSD(n / 10);         // if n = 28   => set_SSD(2)
			toggle_digit(3);         //                on D3

			set_SSD(n % 10);         // 28          => set_SSD(8)
			toggle_digit(4);         //                on D4

			set_SSD(10);
			toggle_digit(2);         // -28
		} else {
			set_SSD(n / 10);         // if n =-28.3  => set_SSD(2)
			toggle_digit(2);         //                 on D2

			set_SSD(n % 10);         // 28           => set_SSD(8)
```

```
                    toggle_digit(3);          //                on D3

                    set_SSD(10);
                    toggle_digit(1);          // -28

                    set_SSD(11);              // -28.
                    toggle_digit(3);

                    set_SSD(f / 100);         //              => set_SSD(3)
                    toggle_digit(4);          // -28.3           on D4
              }
              break;
       case 3:
              set_SSD(n / 100);         // if n = 273   => set_SSD(2)
              toggle_digit(2);          //                 on D2

              set_SSD((n / 10) % 10);   // 27           => set_SSD(7)
              toggle_digit(3);          //                 on D3

              set_SSD(n % 10);          // 273          => set_SSD(3)
              toggle_digit(4);          //                 on D4

              set_SSD(10);
              toggle_digit(1);          // -273
              break;
       }
}
```

```
void display_positive(int n, int f) {
       switch(digit_num(n)) {
       case 1:
              if((f % 10) == 0) {
                     if (((f / 10) % 10) == 0) {
                            if (f == 0) {
                                   set_SSD(n);               // if n = 2     =>
set_SSD(2)
                                   toggle_digit(4);          //
on D4
                            } else {
                                   set_SSD(n);               // if n = 2.10  =>
set_SSD(2)
                                   toggle_digit(3);          //
on D3

                                   set_SSD(11);              // 2.
                                   toggle_digit(3);

                                   set_SSD(f / 100);         // 2.1          =>
set_SSD(1)
                                   toggle_digit(4);          //
on D4
                            }
                     } else {
                            set_SSD(n);               // if n = 2.130 =>
set_SSD(2)
                            toggle_digit(2);          //                 on D2

                            set_SSD(11);              // 2.
                            toggle_digit(2);
```

```
                        set_SSD(f / 100);          // 2.1          =>
set_SSD(1)
                        toggle_digit(3);           //                  on D3

                        set_SSD((f / 10) % 10);    //2.13          =>
set_SSD(2)
                        toggle_digit(4);           //                  on D4
                  }
            }
            else {
                  set_SSD(n);                // if n = 2.132 => set_SSD(2)
                  toggle_digit(1);           //                  on D1

                  set_SSD(11);               // 2.
                  toggle_digit(1);

                  set_SSD(f / 100);          // 2.1          => set_SSD(1)
                  toggle_digit(2);           //                  on D2

                  set_SSD((f / 10) % 10);    // 2.13         => set_SSD(3)
                  toggle_digit(3);           //                  on D3

                  set_SSD(f % 10);           // 2.132        => set_SSD(2)
                  toggle_digit(4);           //                  on D4
            }
            break;
      case 2:
            if (((f / 10) % 10) == 0){
                  if (f == 0) {
                        set_SSD(n / 10);           // if n = 21    =>
set_SSD(2)
                        toggle_digit(3);           //                 on D3

                        set_SSD(n % 10);           // 21             =>
set_SSD(1)
                        toggle_digit(4);           //                 on D4
                  } else {
                        set_SSD(n / 10);           // if n = 21.20 =>
set_SSD(2)
                        toggle_digit(2);           //                  on D2

                        set_SSD(n % 10);           // 21           =>
set_SSD(1)
                        toggle_digit(3);           //                  on D3

                        set_SSD(11);               // 21.
                        toggle_digit(3);

                        set_SSD(f / 100);          // 21.2         =>
set_SSD(2)
                        toggle_digit(4);           //                  on D4
                  }
            } else {
                  set_SSD(n / 10);           // if n = 21.23 => set_SSD(2)
                  toggle_digit(1);           //                  on D1

                  set_SSD(n % 10);           // 21           => set_SSD(1)
                  toggle_digit(2);           //                  on D2
```

```
                  set_SSD(11);              // 21.
                  toggle_digit(2);


                  set_SSD(f / 100);         // 21.2         => set_SSD(2)
                  toggle_digit(3);          //                 on D3

                  set_SSD((f / 10) % 10);   // 21.23        => set_SSD(3)
                  toggle_digit(4);          //                 on D4
            }
            break;
      case 3:
            if (f == 0) {
                  set_SSD(n / 100);         // if n = 212   => set_SSD(2)
                  toggle_digit(2);          //                 on D2

                  set_SSD((n / 10) % 10);   // 21           => set_SSD(1)
                  toggle_digit(3);          //                 on D3

                  set_SSD(n % 10);          // 212          => set_SSD(2)
                  toggle_digit(4);          //                 on D4
            } else {
                  set_SSD(n / 100);         // if n = 212.3 => set_SSD(2)
                  toggle_digit(1);          //                 on D1

                  set_SSD((n / 10) % 10);   // 21           => set_SSD(1)
                  toggle_digit(2);          //                 on D2

                  set_SSD(n % 10);          // 212          => set_SSD(2)
                  toggle_digit(3);          //                 on D3

                  set_SSD(11);              // 212.
                  toggle_digit(3);

                  set_SSD(f / 100);         // 212.3        => set_SSD(3)
                  toggle_digit(4);          //                 on D4
            }
            break;
      case 4:
            set_SSD(n / 1000);          // if n = 2123   => set_SSD(2)
            toggle_digit(1);            //                  on D1

            set_SSD((n / 100) % 10);    // 21            => set_SSD(1)
            toggle_digit(2);            //                  on D2

            set_SSD((n / 10) % 10);     // 212           => set_SSD(2)
            toggle_digit(3);            //                  on D3

            set_SSD(n % 10);            //  2123         => set_SSD(3)
            toggle_digit(4);            //                  on D4
            break;
      }
}
```

```
void display(double num) {
      clear_SSD();

      double in, frac;
      frac = modf(num, &in);
      int n = (int)num;
```

```
	frac *= 1000;                    // if frac     =  0.12345
	int f = abs((int)frac);          // f           =  123

	if (num >= 0) {
		display_positive(n, f);
	}

	if (num < 0) {
		display_negative(n, f);
	}
}
```

```
/* for Keypad - PB4 PB5 PB9 PA8 PB8 PB2 PB0 PA9 */
void set_rows(void) {
	/* set every output 1 */
	GPIOB->ODR  |= (1U << R1); // PB4
	GPIOB->ODR  |= (1U << R2); // PB5
	GPIOB->ODR  |= (1U << R3); // PB9
	GPIOA->ODR  |= (1U << R4); // PA8
}
```

```
void clear_rows(void) {
	/* set every output 0 */
	GPIOB->ODR &= ~(1U << R1); // PB4
	GPIOB->ODR &= ~(1U << R2); // PB5
	GPIOB->ODR &= ~(1U << R3); // PB9
	GPIOA->ODR &= ~(1U << R4); // PA8
}
```