



GEBZE TECHNICAL UNIVERSITY
ELECTRONICS ENGINEERING

ELEC334
MICROPROCESSORS

Randomized Counter

Project 1 Report

Prepared by
1801022022 – Alperen Karataş

1. Introduction

In this project, a counter design was designed that counts backwards with **assembly language** and this counter is displayed with 4-digit seven segment display. The counting process can be controlled with the button and it is also connected with an external LED.

2. Problem

2.1. Problem – Randomized Counter

2.1.1. 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.

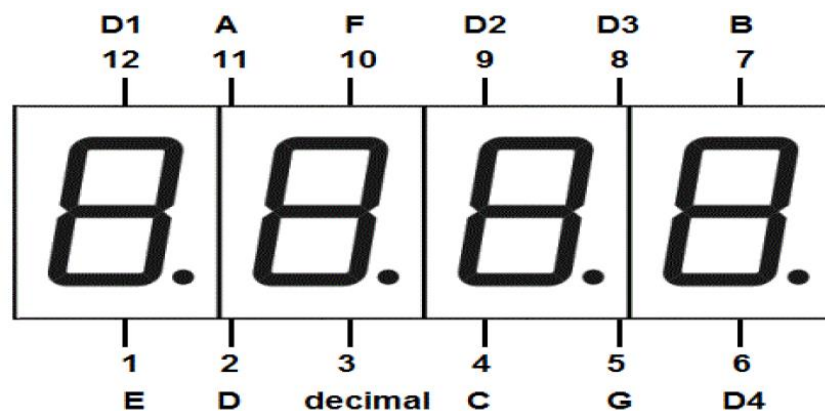


Figure 1. 4-digit Seven Segment Display

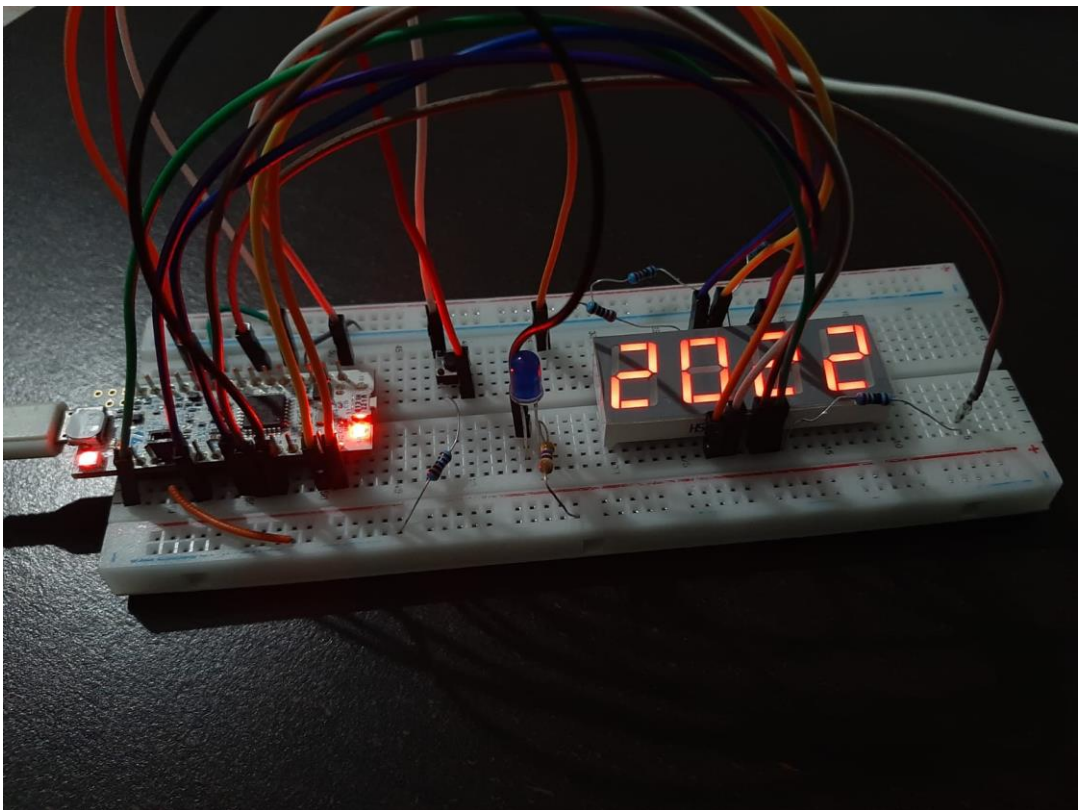
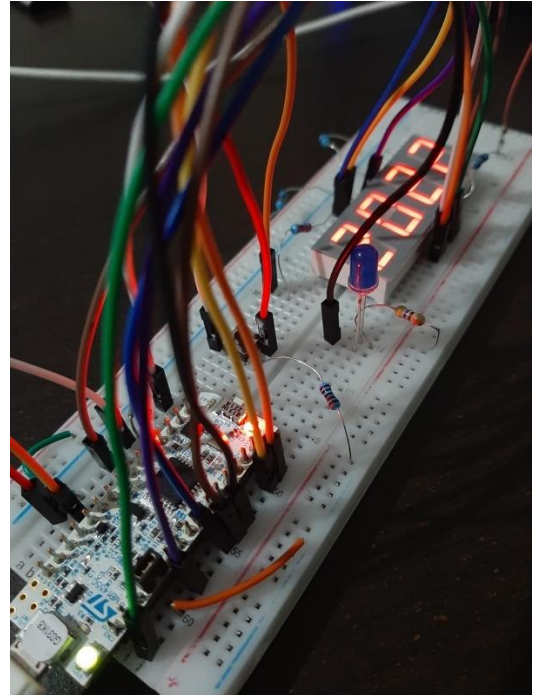
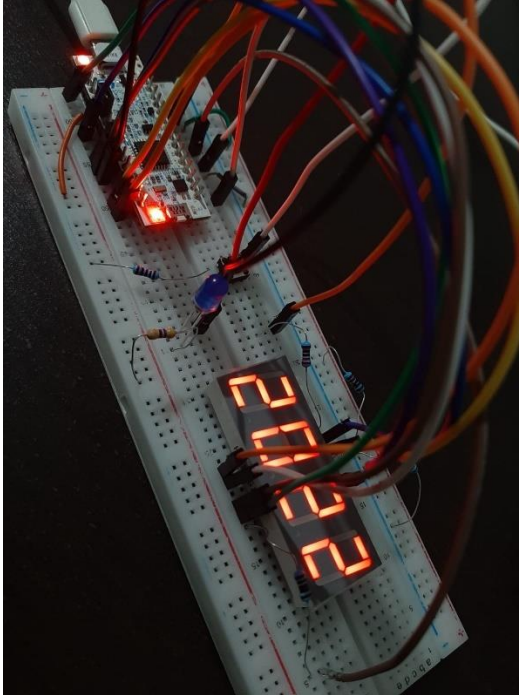


Figure 2. 1-digit Seven Segment Display

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.

2.1.2. Experimental Procedure

Project Setup:

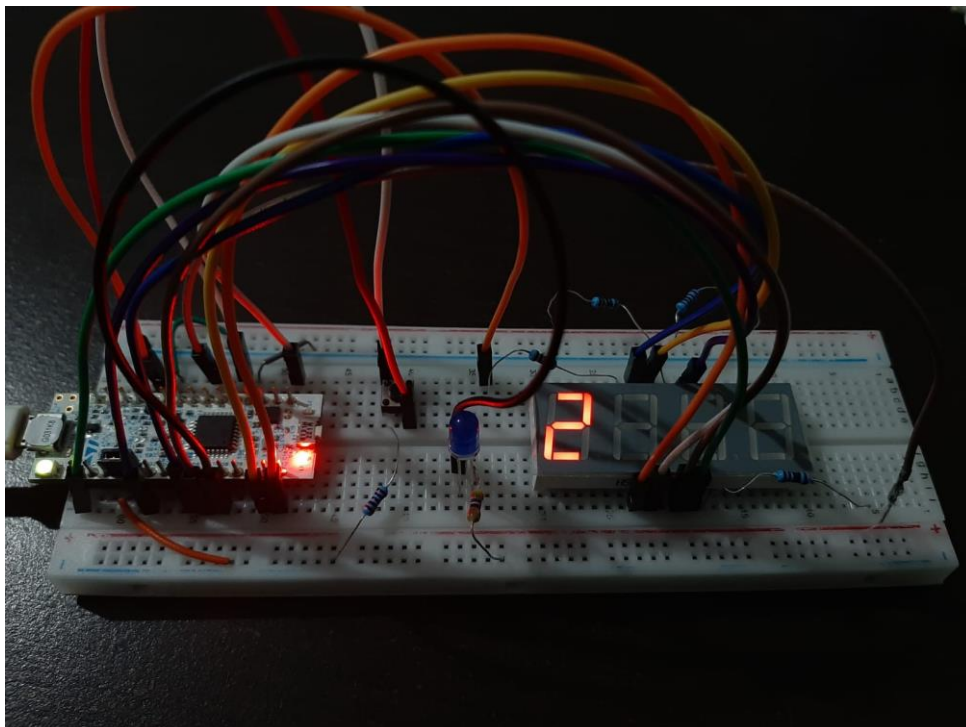


The circuit used in the project setup is as in the photos seen on the previous page. Connections between board and SSD, button and LED has been made by using Figure 1 and Figure 2 are as follows.

A → D6, B → D3, C → D7, D → D13,, E → D12, F → D11,
G → D1, D1 → A7, D1 → A7, D2 → A6, D3 → A0, D4 → A3,
Blue LED → D8 to GND, Push Button → A1 to 5V

Task 1. Light up one digit

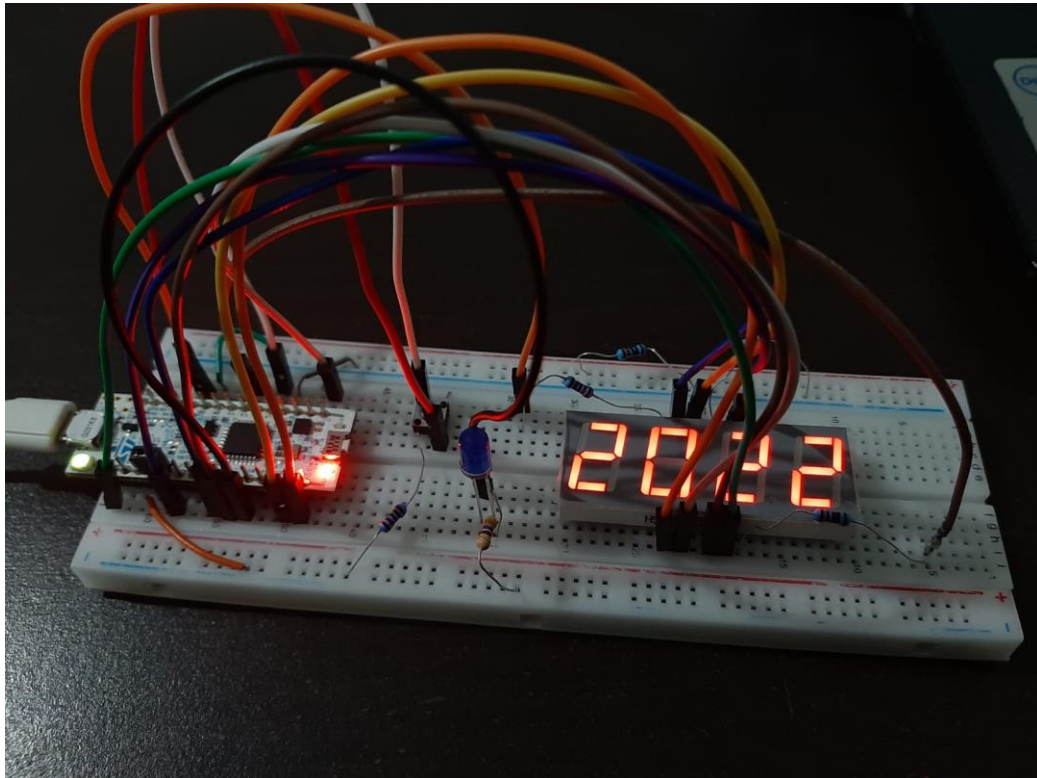
In this task, connections between board and SSD were established using the connections first seen in Figure 1 and Figure 2. D6-D3-D7-D13-D12-D11-D1 connections were made on the board for A-B-C-D-E-F-G LEDs respectively, and PB0-PB1-PB2-PB3-PB4-PB5-PB6 connections were made on the code respectively. On the code, these activated pins are lighted up as desired with simple LED lighting up code. In order for the LEDs to light up, the connecting pin of the relevant digit with the board must be activated. For digit1, the board must activate the A7 pin, that is the PA7 pin via code. It should be remembered that SSD is common anode. In other words, when the desired bit 0 is made, it will light on, and when 1 is made, it will light off. Only when the D1-connected pin is activated will the first digit be active and the desired number will be light on. The following photo shows this situation in the circuit diagram.



Task 2. Light up all digits

As in Task 1, the pin connections mentioned are made in the same way. In order to activate all digits this time, A7-A6-A0-A3 pins must be activated on the board, respectively, which are connected to D1-D2-D3-D4 via SSD. These pins mean activating PA7-PA6-PA0-PA5 pins on the code, respectively. When this is performed, all digits will be active and the desired number will be visible on all digits on the SSD.

The corresponding photo is seen below.



Task 3. Counting

After all digits were activated, counting from the specified number of 9999 to 0 was studied. The first digit is determined as seconds, two, three, and fourth digits milliseconds. Two separate functions have been created on the code for the first digit and for the second, third, fourth digits.

All figures are defined in these functions. In the order in which the code is processed, the top-down counting process takes place.

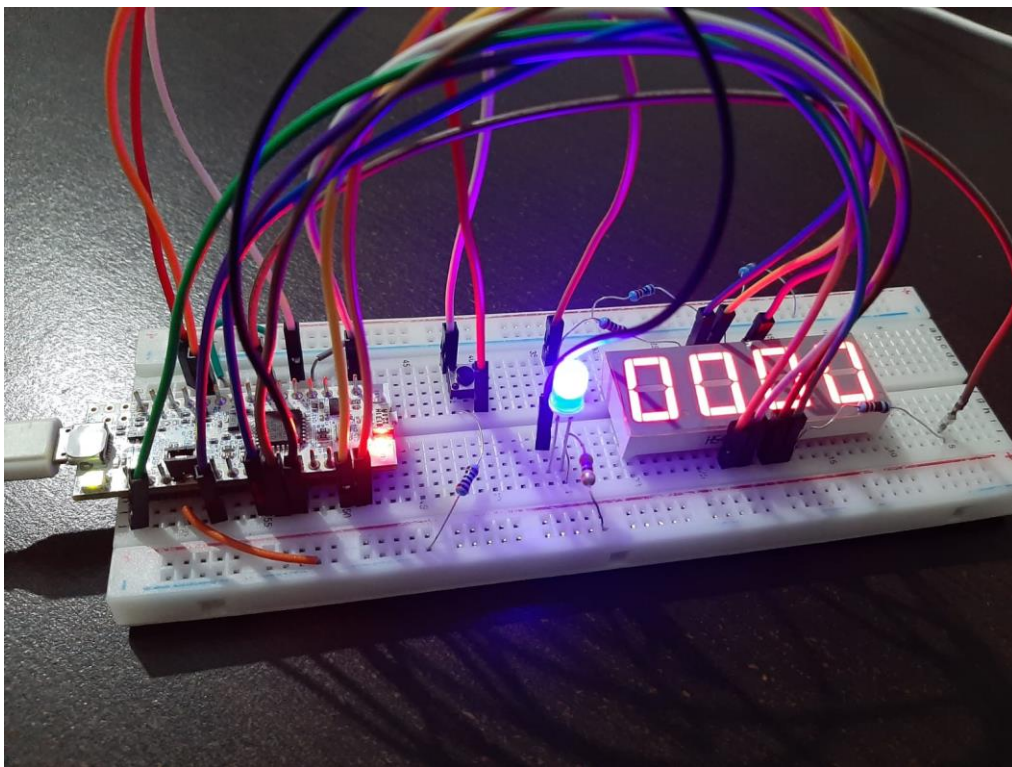
At the end of each second, the function created for other digits is called inside the first digit. Thus, milliseconds can be seen at digits 2,3 and 4 at the end of each second.

The process mentioned can be seen in the YouTube video, which is added with later improvements to the code. These improvements are that the counter starts from the generated random number.

Task 4. Light on the blue LED

In this task, an external LED is added to the circuit. The purpose and task of this LED should be to light on the LED when the counting process is complete, that is, when 0000 is seen on the SSD. The connection of the LED was taken from the D8 pin on the board (PB8 on the code) and carried out by going to the ground with the help of a resistance.

The corresponding photo is seen below.



Task 5. Button Control

The push button is connected to the A1 pin on the board, that is, the PA1 pin on the code. The code is set to start the counter when the button is pressed, and when not, to show the school number, depending on whether the PA1 pin is active.

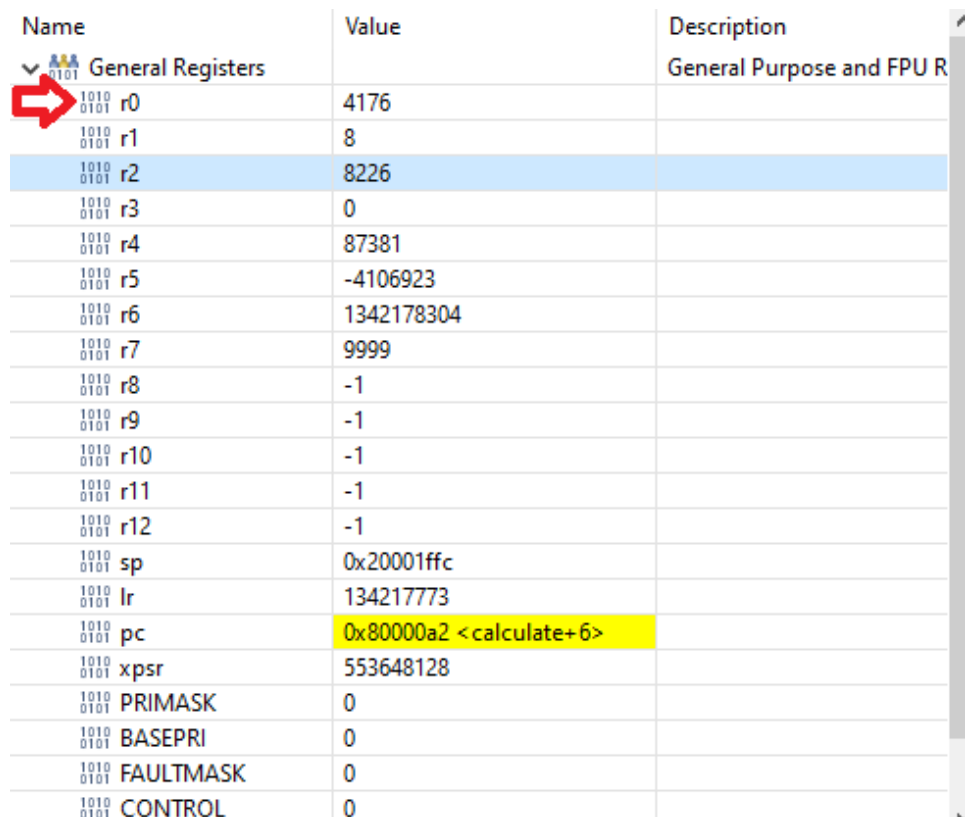
The relevant process can be seen in the added YouTube video.

Task 6. Random number generation

Random number generation was performed using linear congruential generator method. LCG's are known as pseudo random as they require a seed number to generate random number sequence.

This process takes place whose names in **generated_rand_num** and **rand** functions and those that are used in the code. After the code is run, the "step into" process and r0 register status are observed through the program. In **generated_rand_num** function, when the comparison between r0 and r7 registers is complete, the final state of the r0 register is looked at. The number seen is determined as the resulting random number. This number is displayed **loop_d1** function SSD by activating all digits at the beginning of the function written for the first digit, that is, the first digit. After approximately 1 second, the counting process begins.

This register (r0) value obtained on the program is seen in the photo below. (Figure 3)

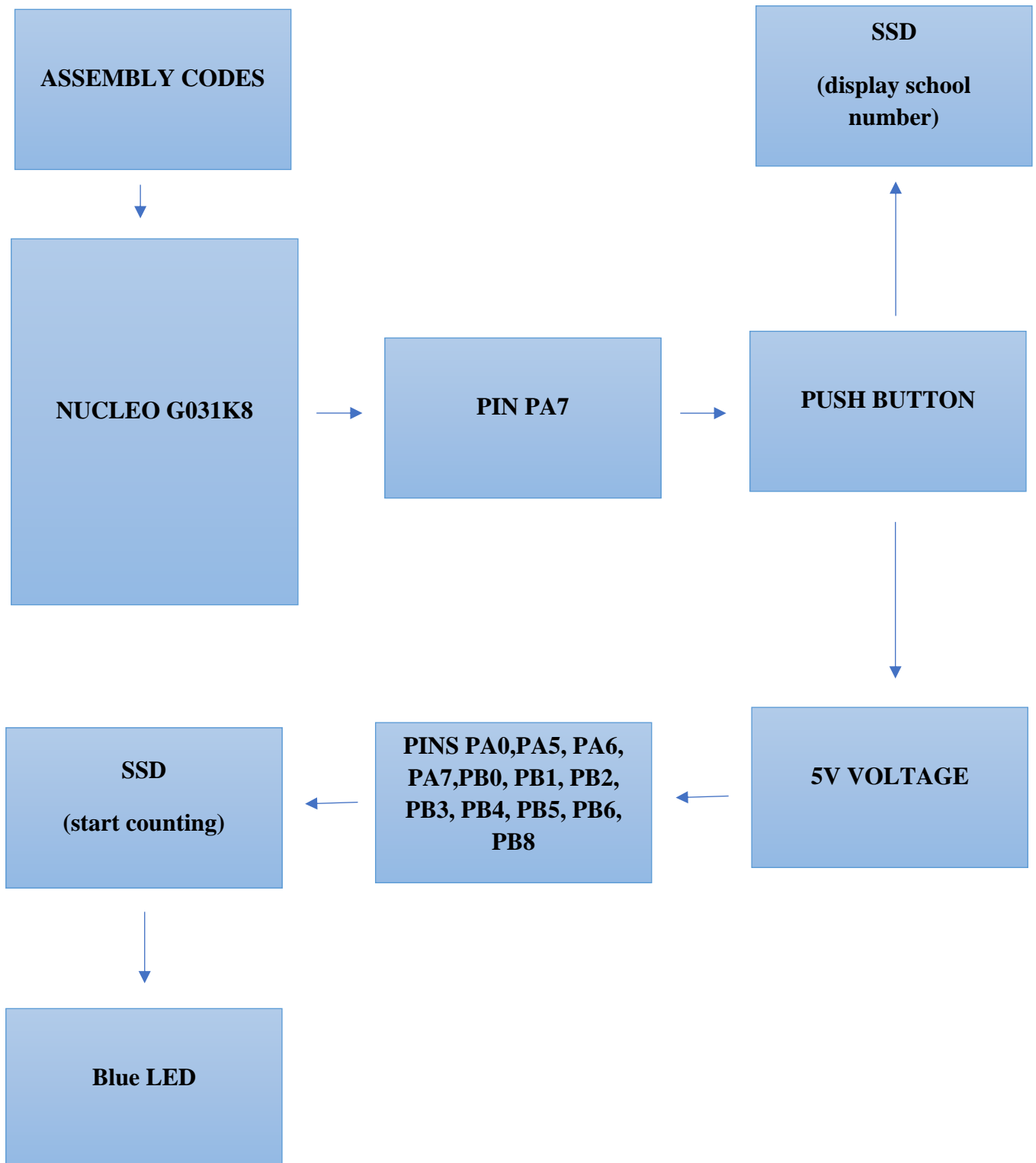


Name	Value	Description
General Registers		General Purpose and FPU R
r0	4176	
r1	8	
r2	8226	
r3	0	
r4	87381	
r5	-4106923	
r6	1342178304	
r7	9999	
r8	-1	
r9	-1	
r10	-1	
r11	-1	
r12	-1	
sp	0x20001ffc	
lr	134217773	
pc	0x80000a2 <calculate+6>	
xpsr	553648128	
PRIMASK	0	
BASEPRI	0	
FAULTMASK	0	
CONTROL	0	

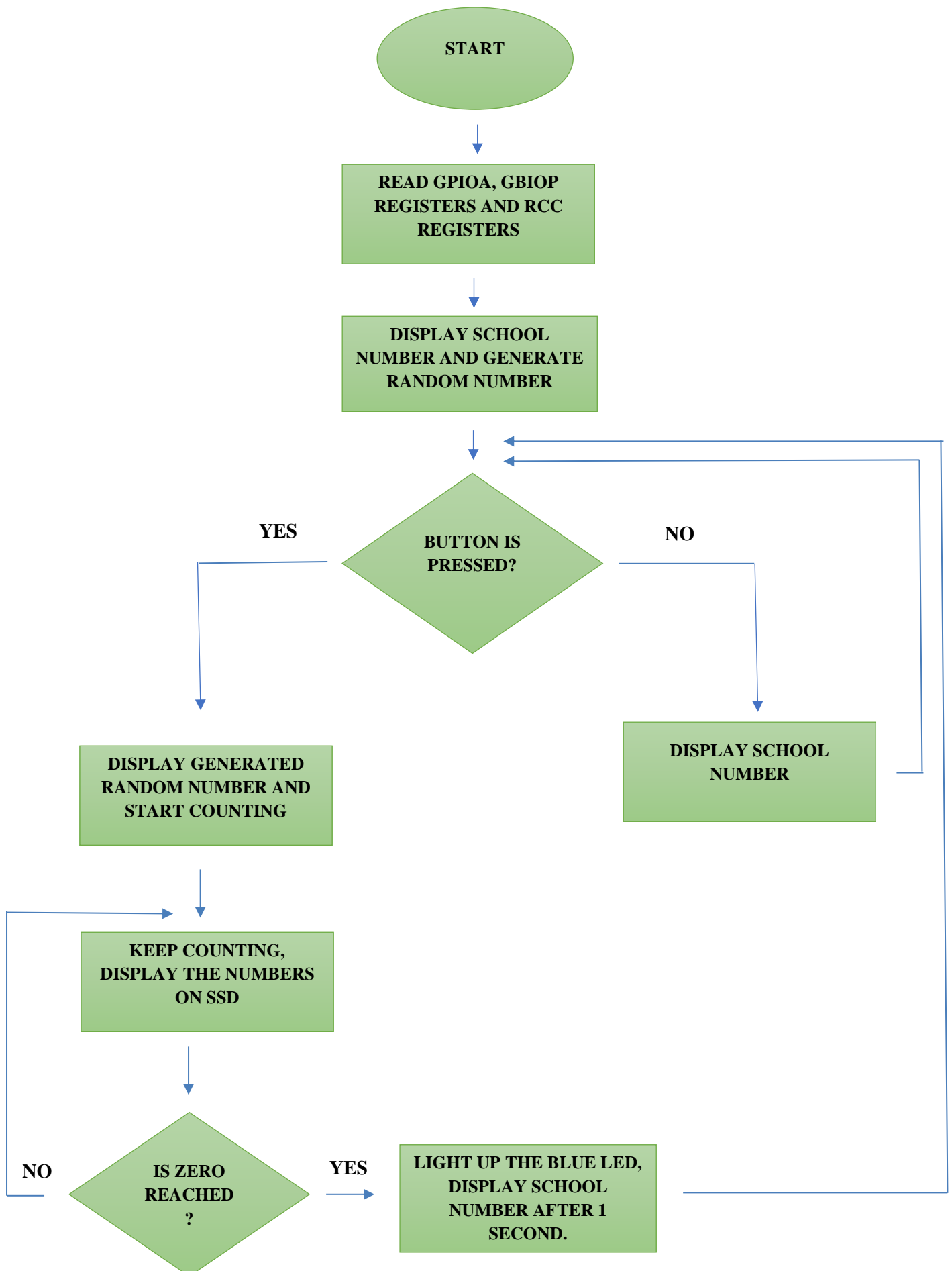
Figure 3. r0 register value

As can be seen from Figure 3, determined random number is 4176.

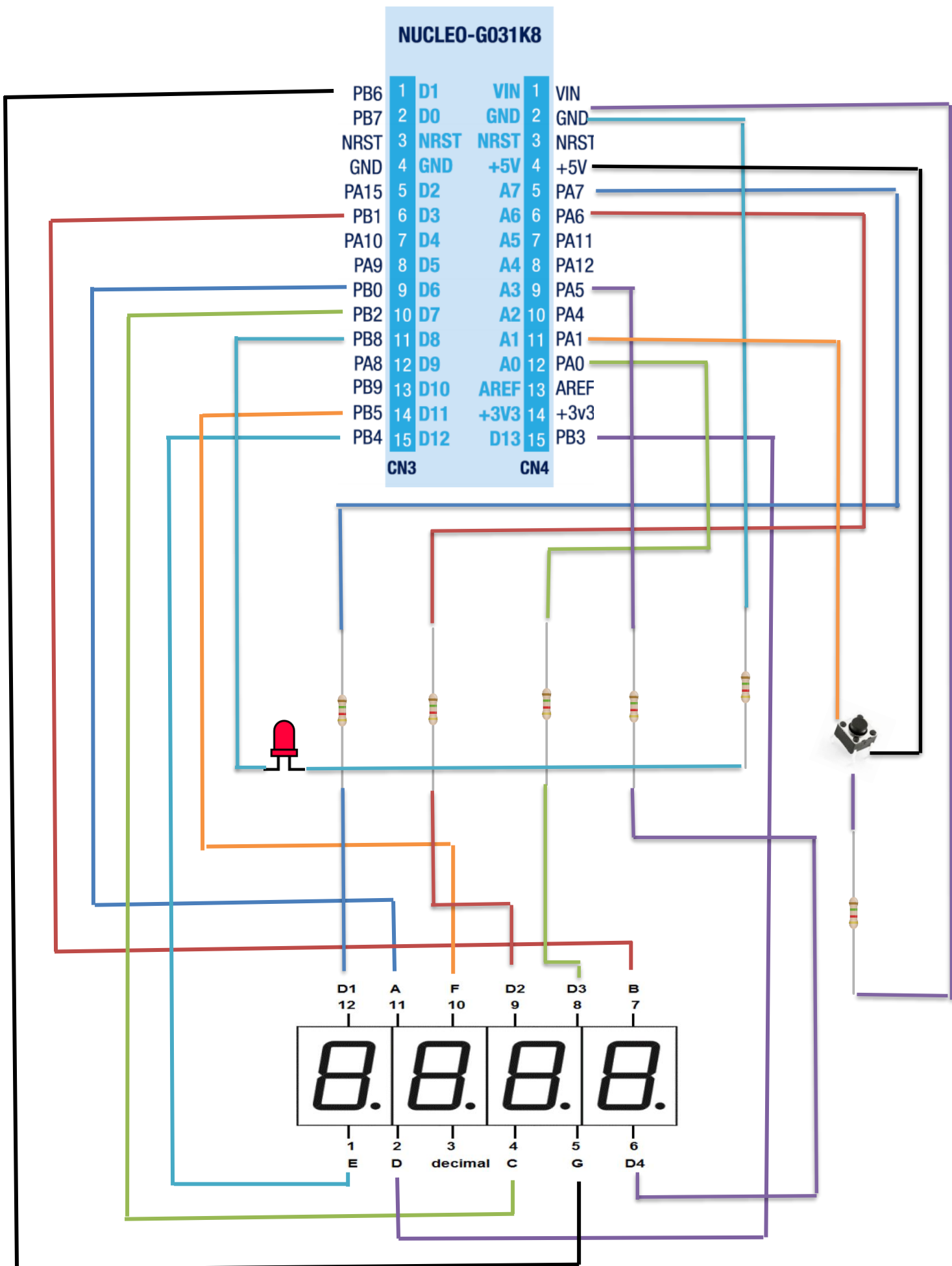
Block Diagram:



Flowchart:



Schematic Diagram:



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

3. Results and General Comments

When all task is complete, the code and circuit work as the way wanted. First, the school number appears on the SSD. Pressing the button generates a random number. Approximately 1 second after this number is generated, the generated random number will begin counting backwards. As soon as 0000 is seen on the SSD, the blue LED will light up and after about 1 second the school number will appear again on the screen. The order in which the task is written is in the same order as the spelling of the code and the establishment of the circuit.

By taking advantage of Figure 1 and Figure 2, the process of light up the desired number on the SSD was not difficult when the relevant connections were established. The difficulty at this stage was to see different numbers in different digits at the same time. To solve this problem, after the desired numbers were stated on the code, one digit was activated at a time within 4 digits, and very small delays were added to the break to ensure continuity. Thus, different numbers were obtained for each digit. This process was necessary for the process of pressing the school number on the screen.

Counting backwards was originally started from the number of 9999. All numbers on the code were defined respectively. With the operation of the code, a decrease was seen as desired. The problem here was to see different digits for different digits at the same time during the reduction process. Even if this problem could not be solved by 100%, it was solved at a certain rate by adding light on-light off operations for the first digit and a certain delay, and much smaller delay times for two, three and fourth digits due to milliseconds.

When the counting process was completed, the LED lightning code was written by connecting to the end of the function written for the first digit for the blue LED lightning process. No problems were encountered here.

After the button was added to the switch, the code phase was started. When the button is pressed, a code is written to start the counting and when not pressed, displaying the school number. The **"bne"** commands **"out of range"** error was a problem that was dealt with for hours in terms of activation of the button. This issue was also resolved by correcting the command **"bne"** to **"b"**. Such a solution was used considering that the command **"b"** goes to the address specified in all circumstances. It was remarkable that only two letters lasted so long.

As desired, a design that stopped counting when the button was pressed again during the counting process could not be done due to the lack of time, the process of solving a problem could be quite long. Although this problem cannot be solved, it is thought that it can be solved when a button control is performed with the **"cmp"** command after the code written for each digit inside the function that performs the counting process.

In Task 6, a number of determinations can be made on the code according to the random number generated. Since the random number is 4176, there is no need to write the numbers 9,8,7,6,5 to the function written for the first digit as the first digit is looked at for the count.

It will certainly be possible to say that it is a very useful project in terms of improving the algorithm structure and solving the problems encountered and approaching the problems with different solution methods. The 4-digit SSD structure was learned very nicely. It became aware of the importance of how to count down, adding delay and determining these delay durations. It was also learned to ensure the coordinated operation of the meter process with the button.

In the light of the results obtained and the comments made, it will be possible to talk about a successful project process in general. What is desired has been largely achieved successfully.

4. Parts List

Row Number	Product name	Piece	Unit Price	Total Price
1	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	6x6 4.2mm Tach Button (4 pin)	1 piece	0,24 TL	0,24 TL
5	5mm blue LED 400-500 mcd	1 piece	0,16 TL	0,16 TL
6	220R resistor	4 pieces	0,03 TL	0,12 TL
7	1K resistor	1 piece	0,03 TL	0,03 TL
8	470R resistor	1 piece	0,045 TL	0,045 TL
9	Male-Male Jumper Cable	15-16 pieces	0,13 TL	2,0181 TL

5. References

- [1] **RM0444 Reference manual** STM32G0x1 advanced Arm®-based 32-bit MCUs
- [2] <https://www.direnc.net/> 14mm 4-digit Common Anode Seven Segment Display Datasheet
- [3] <https://www.hepteknoloji.net/> (Resistor image)
- [4] <https://www.robotistan.com/> (Button image)
- [5] <https://www.sparkfun.com/> (LED image)
- [6] <https://github.com/fcayci>

5. Youtube Links

- [1] <https://youtu.be/qVAUb8v9nhI>

Project Code:

```
/* asm.s
*
* Prepared by:
* Alperen Karataş - 1801022022
*
* Notes:
* ELEC335 2020 Fall Project 1
* Randomized Counter
*
*/

.syntax unified
.cpu cortex-m0plus
.fpu softvfp
.thumb

/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss

/* define peripheral addresses from RM0444 page 57, Tables 3-4 */
.equ RCC_BASE,          (0x40021000)          // RCC base address
.equ RCC_IOPENR,        (RCC_BASE + (0x34)) // RCC IOPENR
register offset

.equ GPIOA_BASE,        (0x50000000)          // GPIOA base address
.equ GPIOA_MODER,       (GPIOA_BASE + (0x00)) // GPIOA MODER
register offset
.equ GPIOA_ODR,         (GPIOA_BASE + (0x14)) // GPIOA ODR register
offset
.equ GPIOB_BASE,        (0x50000400)          // GPIOB base
address
.equ GPIOB_MODER,       (GPIOB_BASE + (0x00)) // GPIOB MODER
register offset
.equ GPIOB_ODR,         (GPIOB_BASE + (0x14)) // GPIOB ODR register
offset
.equ GPIOA_IDR,         (GPIOA_BASE + (0x10)) // GPIOA IDR register
offset
```

```

/* vector table, +1 thumb mode */
.section .vectors
vector_table:
    .word _estack          /* Stack pointer */
    .word Reset_Handler +1 /* Reset handler */
    .word Default_Handler +1 /* NMI handler */
    .word Default_Handler +1 /* HardFault handler */
    /* add rest of them here if needed */

/* reset handler */
.section .text
Reset_Handler:
    /* set stack pointer */
    ldr r0, =_estack
    mov sp, r0

    /* initialize data and bss
     * not necessary for rom only code
     */
    bl init_data
    /* call main */
    bl main
    /* trap if returned */
    b .

/* initialize data and bss sections */
.section .text
init_data:

    /* copy rom to ram */
    ldr r0, =_sdata
    ldr r1, =_edata
    ldr r2, =_sidata
    movs r3, #0
    b LoopCopyDataInit

CopyDataInit:
    ldr r4, [r2, r3]
    str r4, [r0, r3]
    adds r3, r3, #4
LoopCopyDataInit:
    adds r4, r0, r3
    cmp r4, r1
    bcc CopyDataInit
/* zero bss */
    ldr r2, =_sbss
    ldr r4, =_ebss
    movs r3, #0
    b LoopFillZeroBss

```

```
FillZeroBss:
    str r3, [r2]
    adds r2, r2, #4
```

```
LoopFillZeroBss:
    cmp r2, r4
    bcc FillZeroBss
    bx lr
```

```
/* default handler */
```

```
.section .text
```

```
Default_Handler:
```

```
    b Default_Handler
```

```
/* main function */
```

```
.section .text
```

```
main:
```

```
    push {lr}
    /* enable GPIOA clock, bit0 on IOPENR */
    ldr r6, =RCC_IOPENR
    ldr r5, [r6]
    /* movs expects imm8, so this should be fine */
    movs r4, 0x3
    orrs r5, r5, r4
    str r5, [r6]
```

```
    /*enable a lot of the pins for prevent disruption */
```

```
    ldr r6, =GPIOA_MODER
    ldr r5, [r6]
    ldr r4,=# 0x3FFFFFF
    mvns r4,r4
    ands r5,r5,r4
    ldr r4,=# 0x15555
    orrs r5, r5, r4
    str r5,[r6]
```

```
/* setup PA1 for led 00 for bits 2-3 in MODER */
```

```
    ldr r6, =GPIOA_MODER
    ldr r5, [r6]
    /* cannot do with movs, so use pc relative */
    ldr r4, =0x3
    lsls r4, r4, #2
    mvns r4, r4
    ands r5, r5, r4
    str r5, [r6]
```

```

/* enable a lot of the pins for prevent disruption*/
    ldr r6, =GPIOB_MODER
    ldr r5, [r6]
    ldr r4,=# 0x3FFFFFF
    mvns r4,r4
    ands r5,r5,r4
    ldr r4,=# 0x15555
    orrs r5, r5, r4
    str r5,[r6]
    //pop{r1-r7, pc}

/*We can say that the method name is Linear Congruential
Generator.
LCG's are known as pseudo random as they require a seed number
to generate random number sequence.*/
rand:
    ldr r0, =#0x6270      // i choose that
    movs r1, #0x8
    ldr r2, =#0x2022 // i choose that
    ldr r7, =#0x270F
    muls r0, r0, r1
generated_rand_num:
    subs r0, r0, r2
    cmp r7, r0
    blt generated_rand_num
    str r0, [r3]

    ldr r1,= #4000000
    bl delay

/*if button was pressed, start counting*/
/*if button was not pressed, display school number*/
button_ctrl:
    /*ctrl button connected to PA1 in IDR.*/
    ldr r6, =GPIOA_IDR
    ldr r5, [r6]
    lsrs r5, r5, #1
    movs r4, #0x1
    ands r5, r5, r4

    cmp r5, #0x1
    beq loop_d1          // branch to loop_d1 if button was
pressed
    b school_num        // branch to school_num if button was not
pressed

```

```

/*loop_d1 for displaying the number in digit1*/
loop_d1:
    /*enable PA0, PA5, PA6, PA7 pins*/
    ldr r6, =GPIOA_ODR
    ldr r5, [r6]
    ldr r5, =#0x0000
    ldr r3, =#0xE1
    orrs r5, r5, r3
    str r5, [r6]

    ldr r4, = #5000
/*this function using for displaying generated number*/
myrand:
    ldr r6, =GPIOA_ODR
    ldr r5, [r6]
    ldr r5, =#0x0000
    ldr r3, =#0x80
    orrs r5, r5, r3
    str r5, [r6]

    ldr r6, =GPIOB_ODR    // 4
    ldr r5, [r6]
    ldr r3, =#0x19
    ldr r5, =#0x0000
    orrs r5, r5, r3
    str r5, [r6]

/*to ensure continuity, add small delay*/
    ldr r1, = #500
    bl delay

/*light up digit2 connected to PA6; light off digit1-3-4*/
    ldr r6, =GPIOA_ODR
    ldr r5, [r6]
    ldr r5, =#0x0000
    ldr r3, =#0x40
    orrs r5, r5, r3
    str r5, [r6]

    ldr r6, =GPIOB_ODR // 1
    ldr r5, [r6]
    ldr r3, =#0xF9
    ldr r5, =#0x0000
    orrs r5, r5, r3
    str r5, [r6]

/*to ensure continuity, add small delay*/
    ldr r1, = #500
    bl delay

```



```

/*light up digit3 connected to PA0; light off digit1-2-4*/
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x1
orrs r5, r5, r3
str r5, [r6]

ldr r6, =GPIOB_ODR // 7
ldr r5, [r6]
ldr r3, =#0xF8
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

/*to ensure continuity, add small delay*/
ldr r1, = #500
bl delay

/*light up digit4 connected to PA5; light off digit1-2-3*/
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x20
orrs r5, r5, r3
str r5, [r6]

ldr r6, =GPIOB_ODR // 6
ldr r5, [r6]
ldr r3, =#0x02
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

/*to ensure continuity, add small delay*/
ldr r1, = #500
bl delay

subs r4, r4, #1
cmp r4, #0
bne myrand

/* light up digit1 connected to PA7; light off digit2-3-4*/
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x80 // 1000 0000
orrs r5, r5, r3
str r5, [r6]

```

```

/*remember that this is a common anode SSD*/
/*light up the numbers in range (4,0)*/
/* turn on led connected to PB1,PB2,PB5,PB6 in ODR */
ldr r6, =GPIOB_ODR    // 4
ldr r5, [r6]
ldr r3, =#0x19
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

/*wait a bit for prevent flickering as soon as possible*/
ldr r1,= #400000
bl delay

bl loop_d2d3d4

ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x80
orrs r5, r5, r3
str r5, [r6]

/*wait a bit for prevent flickering as soon as possible*/
ldr r1,= #400000
bl delay

/*From here on, the same operations will continue to be applied
for each number (second)*/
/* turn on led connected to PB0,PB1,PB2,PB3,PB6 in ODR */
ldr r6, =GPIOB_ODR // 3
ldr r5, [r6]
ldr r3, =#0xB0
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #400000
bl delay
/*to provide miliseconds, branch to loop_d2d3d4 function for
each numbers (for seconds)*/
bl loop_d2d3d4

ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x80
orrs r5, r5, r3
str r5, [r6]

```

```

ldr r1,= #400000
bl delay

/* turn on led connected to PB0,PB1,PB3,PB4,PB6 in ODR */
ldr r6, =GPIOB_ODR // 2
ldr r5, [r6]
ldr r3, =#0x24
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #400000
bl delay

bl loop_d2d3d4

ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x80
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #400000
bl delay

/* turn on led connected to PB1,PB2 in ODR */
ldr r6, =GPIOB_ODR // 1
ldr r5, [r6]
ldr r3, =#0xF9
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #400000
bl delay

bl loop_d2d3d4

ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x80
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #400000
bl delay

```

```

/* turn on led connected to PB0,PB1,PB2,PB3,PB4,PB5 in ODR */
ldr r6, =GPIOB_ODR // 0
ldr r5, [r6]
ldr r3, =#0x40
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

ldr r1, = #400000
bl delay

bl loop_d2d3d4

/*enable all the digits for displaying 0000*/
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0xE1
orrs r5, r5, r3
str r5, [r6]

/* light up the blue led and get 0000 from SSD. For this,
make 1 the PB8 and PB6 pins. (To get 0000, G led of the SSD which
is connected to PB6, must be 1(common anode).)*/
ldr r6, =GPIOB_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x140
orrs r5, r5, r3
str r5, [r6]

/*display 0000 and wait a bit*/
ldr r1, = #8000000
bl delay

b button_ctrl // branch to button_ctrl function, for
checking the next command which is taken from the board*/

/*For displaying my school number, school_num funciton works*/
school_num:
/* light up digit1 connected to PA7; light off digit2-3-4*/
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x80
orrs r5, r5, r3
str r5, [r6]

```

```

/*my school numbers first digit*/
ldr r6, =GPIOB_ODR // 2
ldr r5, [r6]
ldr r3, =#0x24
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

/*to ensure continuity, add small delay*/
ldr r1, = #500
bl delay

/*light up digit2 connected to PA6; light off digit1-3-4*/
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x40
orrs r5, r5, r3
str r5, [r6]

/*my school numbers second digit*/
ldr r6, =GPIOB_ODR // 0
ldr r5, [r6]
ldr r3, =#0x40
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

/*to ensure continuity, add small delay*/
ldr r1, = #500
bl delay

/*light up digit3 connected to PA0; light off digit1-2-4*/
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x1
orrs r5, r5, r3
str r5, [r6]

/*my school numbers third digit*/
ldr r6, =GPIOB_ODR // 2
ldr r5, [r6]
ldr r3, =#0x24
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

```



```

/*to ensure continuity, add small delay*/
ldr r1,= #500
bl delay

/*light up digit4 connected to PA5; light off digit1-2-3*/
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x20
orrs r5, r5, r3
str r5, [r6]

/*my school numbers fourth digit*/
ldr r6, =GPIOB_ODR // 2
ldr r5, [r6]
ldr r3, =#0x24
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

/*to ensure continuity, add small delay*/
ldr r1,= #500
bl delay

b button_ctrl // branch to button_ctrl function, for
checking the next command which is taken from the board*/

pop {pc}
loop_d2d3d4:
push {r0-r7,lr} // save registers

/*enable digit2-3-4 for miliseconds, and disable digit1*/
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r5, =#0x0000
ldr r3, =#0x61
orrs r5, r5, r3
str r5, [r6]

/*same process which is explained in loop_d1*/
ldr r6, =GPIOB_ODR // 9
ldr r5, [r6]
ldr r3, =#0x10
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #300000
bl delay

```

```

ldr r6, =GPIOB_ODR    // 8
    ldr r5, [r6]
    ldr r3, =#0x00
    ldr r5, =#0x0000
    orrs r5, r5, r3
    str r5, [r6]

    ldr r1, = #300000
    bl delay

    ldr r6, =GPIOB_ODR    // 7
    ldr r5, [r6]
    ldr r3, =#0xF8
    ldr r5, =#0x0000
    orrs r5, r5, r3
    str r5, [r6]

    ldr r1, = #300000
    bl delay

    ldr r6, =GPIOB_ODR // 6
    ldr r5, [r6]
    ldr r3, =#0x02
    ldr r5, =#0x0000
    orrs r5, r5, r3
    str r5, [r6]

    ldr r1, = #300000
    bl delay

ldr r6, =GPIOB_ODR    // 5
    ldr r5, [r6]
    ldr r3, =#0x12
    ldr r5, =#0x0000
    orrs r5, r5, r3
    str r5, [r6]

    ldr r1, = #300000
    bl delay

    ldr r6, =GPIOB_ODR    // 4
    ldr r5, [r6]
    ldr r3, =#0x19
    ldr r5, =#0x0000
    orrs r5, r5, r3
    str r5, [r6]

```

```

ldr r1,= #300000
bl delay

ldr r6, =GPIOB_ODR // 3
ldr r5, [r6]
ldr r3, =#0xB0
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #300000
bl delay

ldr r6, =GPIOB_ODR // 2
ldr r5, [r6]
ldr r3, =#0x24
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #300000
bl delay

ldr r6, =GPIOB_ODR // 1
ldr r5, [r6]
ldr r3, =#0xF9
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #300000
bl delay

ldr r6, =GPIOB_ODR // 0
ldr r5, [r6]
ldr r3, =#0x40
ldr r5, =#0x0000
orrs r5, r5, r3
str r5, [r6]

ldr r1,= #300000
bl delay

pop {r0-r7,pc} // save registers and program counter to
continue from loop_d1

```

delay:

```
subs r1,r1,#1  
bne delay  
bx lr
```

```
/* for(;;); */  
b .
```

```
/* this should never get executed */  
nop
```