GEBZE TECHNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC 334

Microprocessors Lab

MIDTERM #1

Randomized Counter

Prepared by
1) 1801022037 Ömer Emre POLAT

1. Introduction

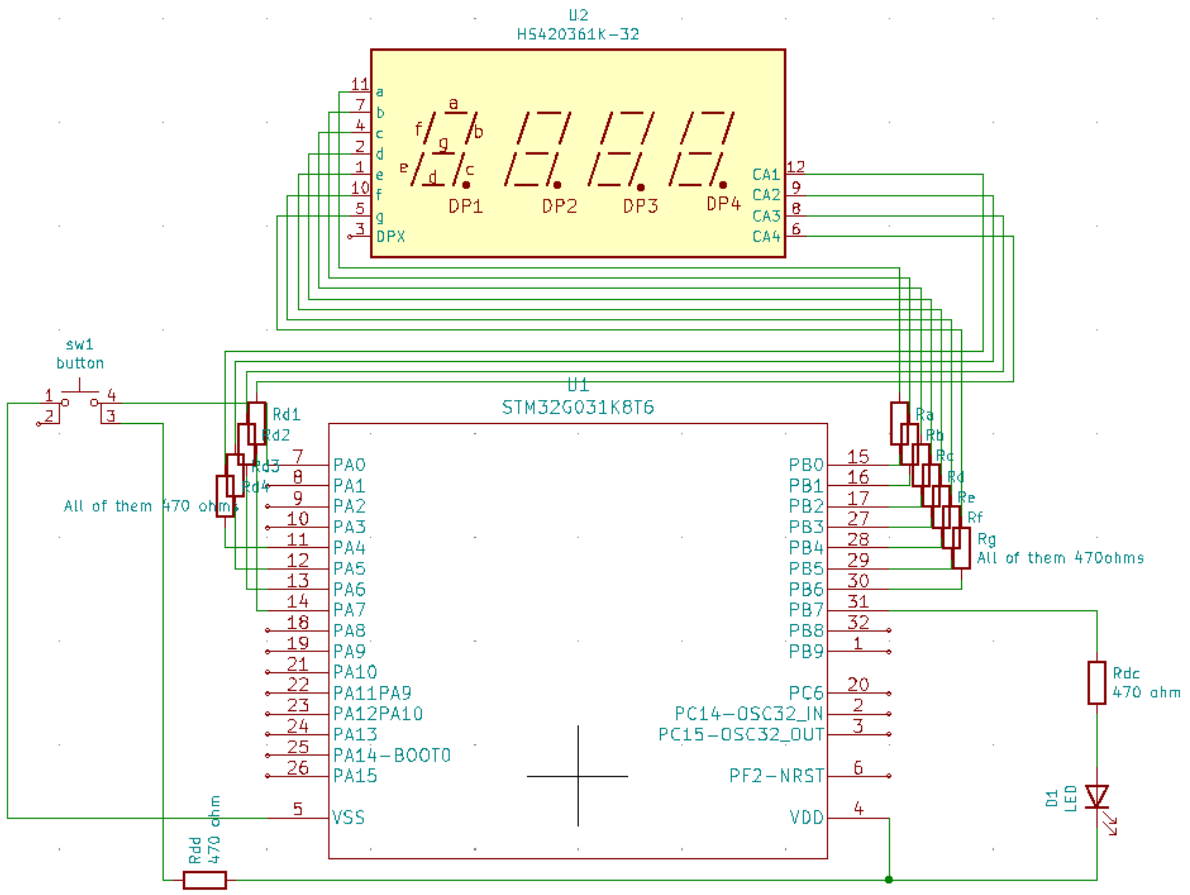
In this Project we will make a randomized counter on the stm32g031k8t6 development board. This Project will use a seven-segment display and an external LED as the peripherals and design a randomized counter with the given design restrictions.

2. Project Design

2.1. Flowchart and Block Diagram

First we will design the board connections and the flowchart for our code for ease of design.

The board connections will be drawn on KiCAD using the model libraries of the components.



All the resistors seen in the figure are 470 ohm resistors.

The pin connections are as follows:

PB0 -> A (SSD)

PB1 -> B (SSD)

....

PB6 -> G (SSD)

PB7 -> External LED

PA4 -> D1 (SSD)

PA5 -> D2 (SSD)

PA6 -> D3 (SSD)

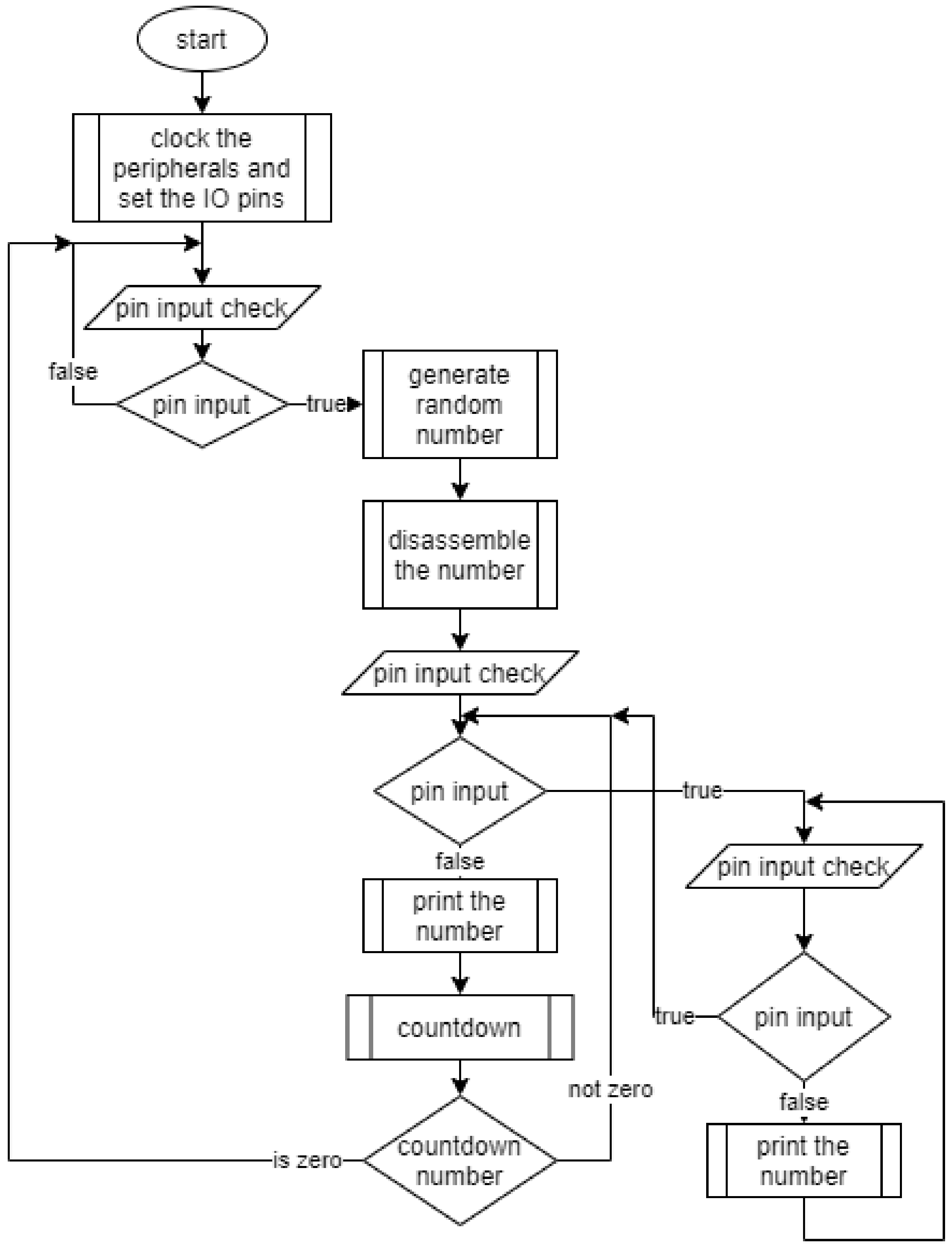
PA7 -> D4 (SSD)

PA0 -> Switch Input Pin

Now that the pins are chosen we can start designing the flowchart and the code.

Code will first set the peripherals and pin input output states. The runtime code will start with the idle state where the number is shown. After the button is pressed it will start with the generation of a pseudo random number with the built-in seed. This seed will be stored in the stack for fast access.

After the random number is generated the random number will be disassembled into 4 digits and the pins will light up according to those digits. The countdown will begin and count the randomly generated number up to zero. Button check will be done when on idle state and on countdown state. If the button is pressed on the countdown state it will go into the countdown halt state where it will print the same number until the button is pressed again.

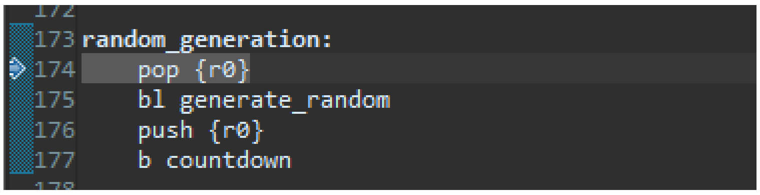


The LED and the delays will be added the written code later. Now that the Flowchart is created, the code can be written in arm. First the code will be tested on STM32 IDE with no peripherals connected to the board to see if the code is working as intended.

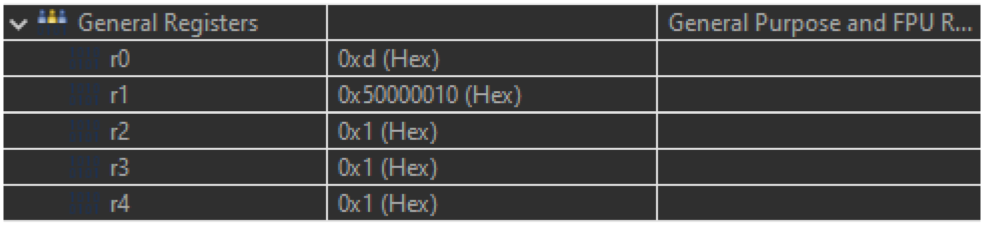
2.2 Writing the Random Function

The random function will be implemented with the same algorithm used on the first pseudo-random number generator written on C. As Pseudo-random number generation algorithm, Linear Congruential Generator will be used since it is short and easier to implement on a low level language like assembly.

```
generate_random:

    push {lr}

    ldr r1, =0x41A7 /* a */
    ldr r2, =0x3039 /* c */
    ldr r3, =0x7FFF /* m */

    muls r0, r1, r0 /* seed = ((seed * a) + c) % m */
    adds r0, r0, r2
    movs r4, r3
    bl modulus

    ldr r4, =0x2327 /* seed = seed % 8999 */
    bl modulus

    ldr r5, =0x3E8 /* temporary shift to 1000 - 9999 constant */
    adds r0, r0, r5
    pop {pc}

/* /////////////////////////////////////////////////////////////////// */

modulus: /* uses r0 and r4 for computation of r0 % r4 */

    push {lr}

modulus_loop:

    cmp r0, r4
    blt pop_pc
    subs r0, r0, r4 /* subtract r0 from r4*/
    /* compare if the r0 value is lower than r4 if so jump out of division */
    bge modulus_loop /* jump back*/
    pop {pc} /*
```

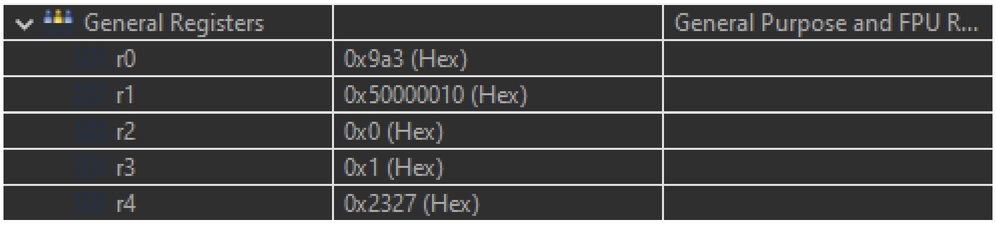


```
random_generation:
    pop {r0}
    bl generate_random
    push {r0}
    b countdown
```



General Registers		General Purpose and FPU R...
r0	0xd (Hex)	
r1	0x50000010 (Hex)	
r2	0x1 (Hex)	
r3	0x1 (Hex)	
r4	0x1 (Hex)	

After the code is run the registers look like the table given below.



General Registers		General Purpose and FPU R...
r0	0x9a3 (Hex)	
r1	0x50000010 (Hex)	
r2	0x0 (Hex)	
r3	0x1 (Hex)	
r4	0x2327 (Hex)	

The random number generation code created the random number 2467 as its first random number. When we run the code multiple times we get the results of:

```
2467, 7961, 8213, 1,865
```

Now that we have created the random generation code we can write up the number extraction code which will disassemble the number into its digits and store them in the r1, r2, r3 and r4 registers.

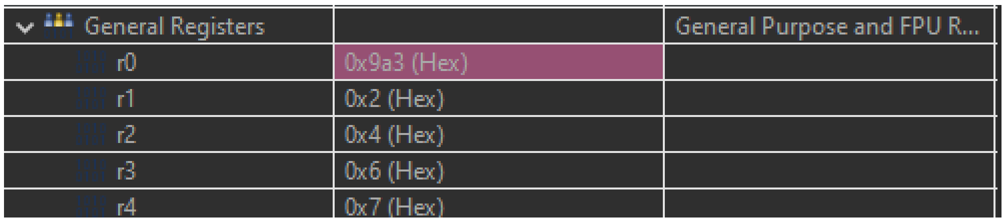
2.3 Number Extraction

Number extraction code is written by substraction 1000, 100 and 10s from the number and counting them.

```
number_extraction:

    push {lr} /* pushes lr to return back with pop {pc} */
    push {r0} /* pushes the seed into the stack to preserve its value */
    movs r1, #0x0
    movs r2, #0x0
    movs r3, #0x0
    movs r4, #0x0 /* resetting the registers for counting */

    ldr r5, =0x3E8 /* dec: 1000 */

thousands_loop:

    cmp r0, r5 /* removing 1000s until the number is smaller than thousand and
```

```
 counting it to r1 */
    blt hundreds_loop
    subs r0, r0, r5
    adds r1, #0x1
    b thousands_loop

hundreds_loop:

    cmp r0, #0x64 /* removing 100s until the number is smaller than thousand a
nd counting it to r1 */
    blt tens_loop
    subs r0, #0x64
    adds r2, #0x1
    b hundreds_loop

tens_loop:

    cmp r0, #0xA /* removing 10s until the number is smaller than thousand and
 counting it to r1 */
    blt ones_loop
    subs r0, #0xA
    adds r3, #0x1
    b tens_loop

ones_loop:

    movs r4, r0 /* writing the left value to the ones digit register */
    pop {r0}
    b pop_pc
```

The code is commented with the descriptions of what parts do what operation. Now we test the code on STM32 IDE and check if it works.



General Registers		General Purpose and FPU R...
r0	0x9a3 (Hex)	
r1	0x2 (Hex)	
r2	0x4 (Hex)	
r3	0x6 (Hex)	
r4	0x7 (Hex)	

As we can see the number 0x9A3 is disassembled into 2647 which is the correct number. With this part done we can move into the number printing functions.

2.4 Number Printing Functions

Number printing function will work by using other sub functions that turn the output pins according to light up a certain number. This will be done with multiple set_number functions. Writing the functions is pretty simple but they have to work as a subfunction to the main function that chooses which set function to run. There will be set_D1, set_D2… functions aswell to set the digit to write the number. Lastly there will be a set_none function to turn off all the pins so that when the chosen digit is switching the number wont pop up on the other digit.

```
set_D1:

    push {lr}
    ldr r7, =0x50000014 /* sets the D1 pin high */

    ldr r6, =0x0010
    str r6, [r7]
    pop {pc}

set_D2:

    push {lr}
    ldr r7, =0x50000014 /* sets the D2 pin high */

    ldr r6, =0x0020
    str r6, [r7]
    pop {pc}

set_D3:

    push {lr}
    ldr r7, =0x50000014 /* sets the D3 pin high */

    ldr r6, =0x0040
    str r6, [r7]
    pop {pc}

set_D4:

    push {lr}
    ldr r7, =0x50000014 /* sets the D4 pin high */

    ldr r6, =0x0080
    str r6, [r7]
    pop {pc}

/* /////////////////////NUMBER FUNCTIONS/////////////////////////// */

set_none:

    push {lr}
    ldr r7, =0x50000414 /* sets all the A,B,C...G pins to high to turn all of
them off */
```

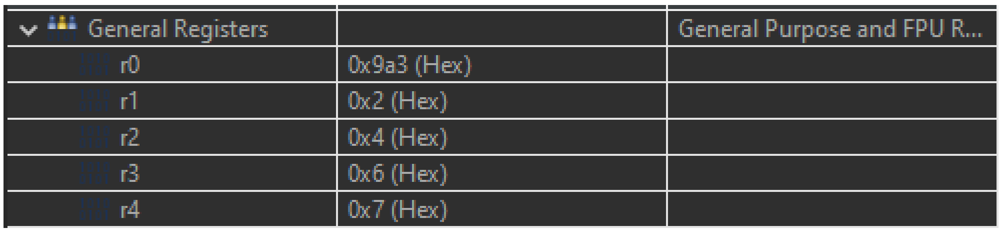
```
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0xFF
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_0:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 0 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0xC0
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_1:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 1 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0xF9
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_2:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 2 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0xA4
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_3:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 3 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0xB0
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_4:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 4 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0x99
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_5:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 5 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0x92
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_6:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 6 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0x82
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_7:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 7 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0xF8
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_8:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 8 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0x80
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}

set_9:

    push {lr}
    ldr r7, =0x50000414 /* sets the A,B,C...G pins to show the number 9 */
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0x90
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}
```

These sub functions will help writing the main function that picks the correct functions to run when printing the number to the SSD (seven segment display). Now we can write the driver function to use these sub functions.

```
number_picker:

    push {lr} /* pushes lr so that we can use it as a subfunction */

thousands_picker:

    bl set_none /* clears the written number */
    bl set_D1 /* sets the digit to the first one */
    cmp r1, #0x0 /* check the thousands digit r1 and runs the according write
function */
    beq run_thousands_0
    cmp r1, #0x1
    beq run_thousands_1
    cmp r1, #0x2
    beq run_thousands_2
    cmp r1, #0x3
    beq run_thousands_3
    cmp r1, #0x4
    beq run_thousands_4
    cmp r1, #0x5
    beq run_thousands_5
    cmp r1, #0x6
    beq run_thousands_6
    cmp r1, #0x7
    beq run_thousands_7
    cmp r1, #0x8
    beq run_thousands_8
    cmp r1, #0x9
    beq run_thousands_9

run_thousands_0:

    bl set_0
    b hundreds_picker

run_thousands_1:

    bl set_1
    b hundreds_picker

run_thousands_2:

    bl set_2
    b hundreds_picker

run_thousands_3:

    bl set_3
    b hundreds_picker

run_thousands_4:

    bl set_4
    b hundreds_picker

run_thousands_5:

    bl set_5
    b hundreds_picker

run_thousands_6:
    bl set_6
    b hundreds_picker

run_thousands_7:

    bl set_7
    b hundreds_picker

run_thousands_8:

    bl set_8
    b hundreds_picker

run_thousands_9:

    bl set_9
    b hundreds_picker

hundreds_picker:

    bl set_none /* clears the written number */
    bl set_D2 /* sets the digit to the second one */
    cmp r2, #0x0 /* check the hundreds digit r2 and runs the according write f
unction */
    beq run_hundreds_0
    cmp r2, #0x1
    beq run_hundreds_1
    cmp r2, #0x2
    beq run_hundreds_2
    cmp r2, #0x3
    beq run_hundreds_3
    cmp r2, #0x4
    beq run_hundreds_4
    cmp r2, #0x5
    beq run_hundreds_5
    cmp r2, #0x6
    beq run_hundreds_6
    cmp r2, #0x7
    beq run_hundreds_7
    cmp r2, #0x8
    beq run_hundreds_8
    cmp r2, #0x9
    beq run_hundreds_9

run_hundreds_0:

    bl set_0
    b tens_picker
```

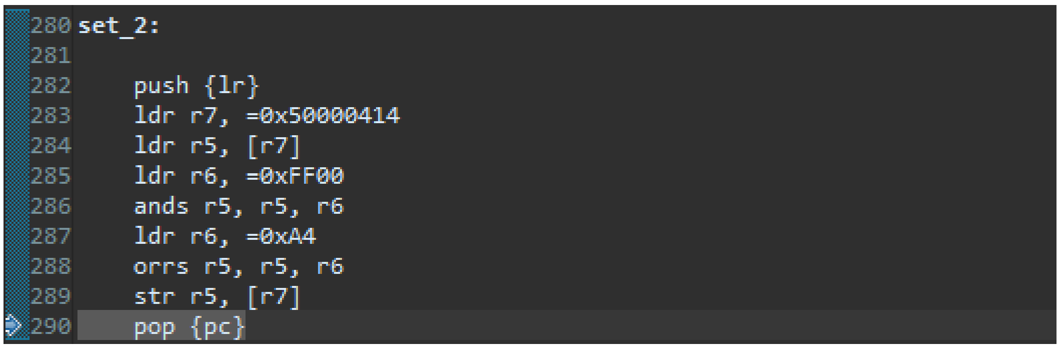
```
run_hundreds_1:

    bl set_1
    b tens_picker

run_hundreds_2:

    bl set_2
    b tens_picker

run_hundreds_3:

    bl set_3
    b tens_picker

run_hundreds_4:

    bl set_4
    b tens_picker

run_hundreds_5:

    bl set_5
    b tens_picker

run_hundreds_6:

    bl set_6
    b tens_picker

run_hundreds_7:

    bl set_7
    b tens_picker

run_hundreds_8:

    bl set_8
    b tens_picker

run_hundreds_9:

    bl set_9
    b tens_picker

tens_picker:

    bl set_none /* clears the written number */
    bl set_D3 /* sets the digit to the third one */
    cmp r3, #0x0 /* check the tens digit r3 and runs the according write funct
ion */
    beq run_tens_0
    cmp r3, #0x1
    beq run_tens_1
    cmp r3, #0x2
    beq run_tens_2
    cmp r3, #0x3
    beq run_tens_3
    cmp r3, #0x4
    beq run_tens_4
    cmp r3, #0x5
    beq run_tens_5
    cmp r3, #0x6
    beq run_tens_6
    cmp r3, #0x7
    beq run_tens_7
    cmp r3, #0x8
    beq run_tens_8
    cmp r3, #0x9
    beq run_tens_9

run_tens_0:

    bl set_0
    b ones_picker

run_tens_1:

    bl set_1
    b ones_picker

run_tens_2:

    bl set_2
    b ones_picker

run_tens_3:

    bl set_3
    b ones_picker

run_tens_4:

    bl set_4
    b ones_picker

run_tens_5:
    bl set_5
    b ones_picker

run_tens_6:

    bl set_6
    b ones_picker

run_tens_7:

    bl set_7
    b ones_picker

run_tens_8:

    bl set_8
    b ones_picker

run_tens_9:

    bl set_9
    b ones_picker

ones_picker:

    bl set_none /* clears the written number */
    bl set_D4 /* sets the digit to the third one */
    cmp r4, #0x1 /* check the tens digit r3 and runs the according write funct
ion */
    beq run_ones_1
    cmp r4, #0x2
    beq run_ones_2
    cmp r4, #0x3
    beq run_ones_3
    cmp r4, #0x4
    beq run_ones_4
    cmp r4, #0x5
    beq run_ones_5
    cmp r4, #0x6
    beq run_ones_6
    cmp r4, #0x7
    beq run_ones_7
    cmp r4, #0x8
    beq run_ones_8
    cmp r4, #0x9
    beq run_ones_9

run_ones_0:
    bl set_0
    b pop_pc

run_ones_1:

    bl set_1
    b pop_pc

run_ones_2:

    bl set_2
    b pop_pc

run_ones_3:

    bl set_3
    b pop_pc

run_ones_4:

    bl set_4
    b pop_pc

run_ones_5:

    bl set_5
    b pop_pc

run_ones_6:

    bl set_6
    b pop_pc

run_ones_7:

    bl set_7
    b pop_pc

run_ones_8:

    bl set_8
    b pop_pc

run_ones_9:

    bl set_9
    b pop_pc
```

Now that the number picker is written we can test the code using all the functions before as the test bench.



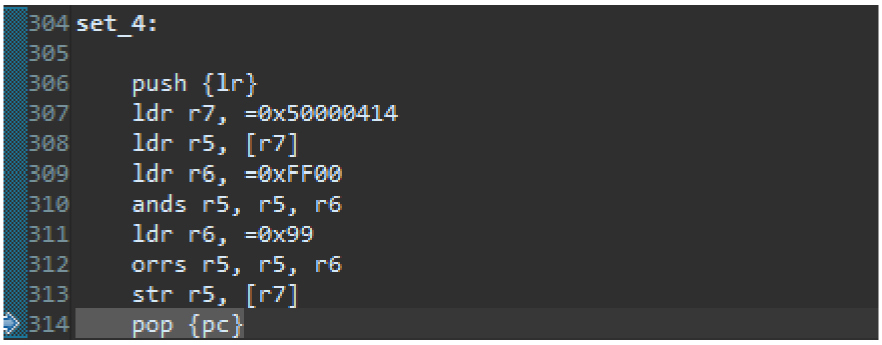
General Registers		General Purpose and FPU R...
r0	0x9a3 (Hex)	
r1	0x2 (Hex)	
r2	0x4 (Hex)	
r3	0x6 (Hex)	
r4	0x7 (Hex)	

With the input variables set by the random generation and number extraction functions the picker function can pick the according functions to run.



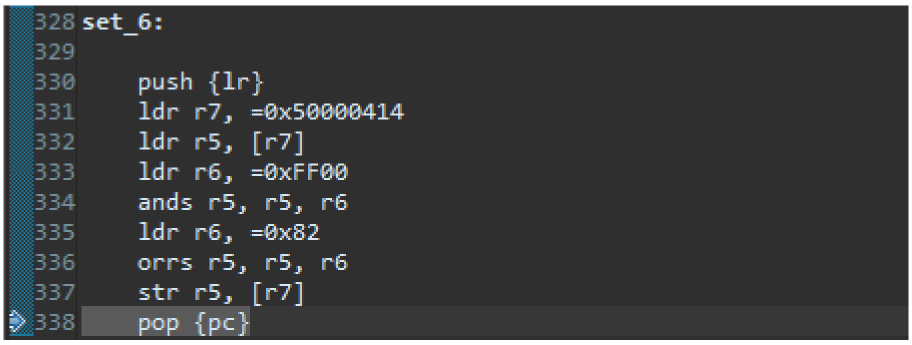
```
set_2:

    push {lr}
    ldr r7, =0x50000414
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0xA4
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}
```

It chooses set_2 as the thousands digit which is correct.



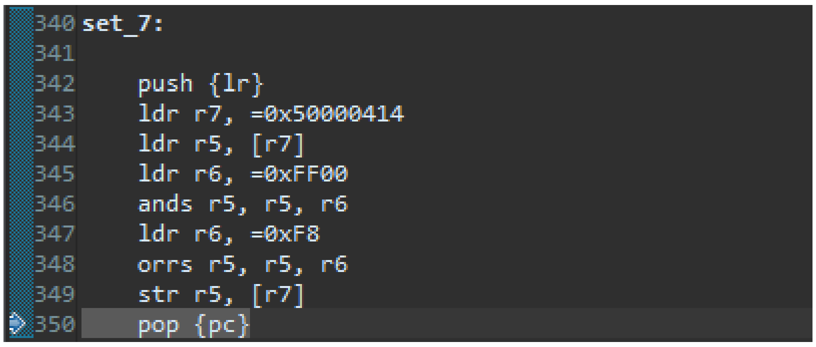
```
set_4:

    push {lr}
    ldr r7, =0x50000414
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0x99
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}
```

It chooses set_4 as the hundreds digit which is correct.



```
set_6:

    push {lr}
    ldr r7, =0x50000414
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0x82
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}
```

It chooses set_6 as the tesn digit which is correct.



```
set_7:

    push {lr}
    ldr r7, =0x50000414
    ldr r5, [r7]
    ldr r6, =0xFF00
    ands r5, r5, r6
    ldr r6, =0xF8
    orrs r5, r5, r6
    str r5, [r7]
    pop {pc}
```

At last it chooses set_7 as the last ones digit which is again correct.

After checking the output pins with a multimeter we can confirm if the pins are given in the correct way. After checking it we can plug the peripherals and check if the seven segment display works.

With the peripherals connected the test is successful and we can write the generated random number to individual digits with no problems. Now that the code works we can start creating the main code that runs these functions according to design requirements.

2.5 Main Code

Main code will be written with the design restrictions and the refresh rate of the seven segment display. With these in mind we can create the main code using all the before functions.

```
/* //////ENABLE GPIOA CLOCK////// */
    ldr r1, =(0x40021034) /* RCC with IOPENR offset */
    ldr r0, [r1]

    ldr r2, =0x1 /* A port enable */
    orrs r0, r0, r2 /* change the corresponding bit to 1 */
    str r0, [r1] /* enable GPIOA clock */

    /* //////ENABLE GPIOB CLOCK////// */
    ldr r1, =(0x40021034) /* RCC with IOPENR offset */
    ldr r0, [r1]

    ldr r2, =0x2 /* for B port */
    orrs r0, r0, r2 /* change the corresponding bit to 1 */
    str r0, [r1] /* enable GPIOB clock */

    /* //////SETUP A PINS MODER////// */
    ldr r1, =(0x50000000)
    ldr r0, [r1]

    ldr r2, =0xFFFFFFFC /* set A0 pin to input */
    ands r0, r0, r2
    ldr r2, =0xFFFF55FF /* set A4-7 pins to output */
    ands r0, r0, r2

    ldr r2, =0x5500
    orrs r0, r0, r2
    str r0, [r1] /* writing back the modified MODER bits */

    /* //////SETUP B PINS MODER////// */
    ldr r1, =(0x50000400)
    ldr r0, [r1]

    ldr r2, =0xFFFF5555 /* set B0-7 pins to output */
    ands r0, r0, r2
    ldr r2, =0x5555
    orrs r0, r0, r2
    str r0, [r1] /* writing back the modified MODER bits */

    /* //////SETUP VARIABLES////// */
    ldr r0, =0xD /*seed*/

    /* /////RUNTIME CODE/////// */

    push {r0}

idle_loop:

    bl get_button
    cmp r2, r3
    beq random_generation

    ldr r6, =0x0010
    bl set_D
    bl set_2
    bl set_none
    ldr r6, =0x0020
    bl set_D
    bl set_0
    bl set_none
    ldr r6, =0x0040
    bl set_D
    bl set_3
    bl set_none
    ldr r6, =0x0080
    bl set_D
    bl set_7
    bl set_none

    b idle_loop
```

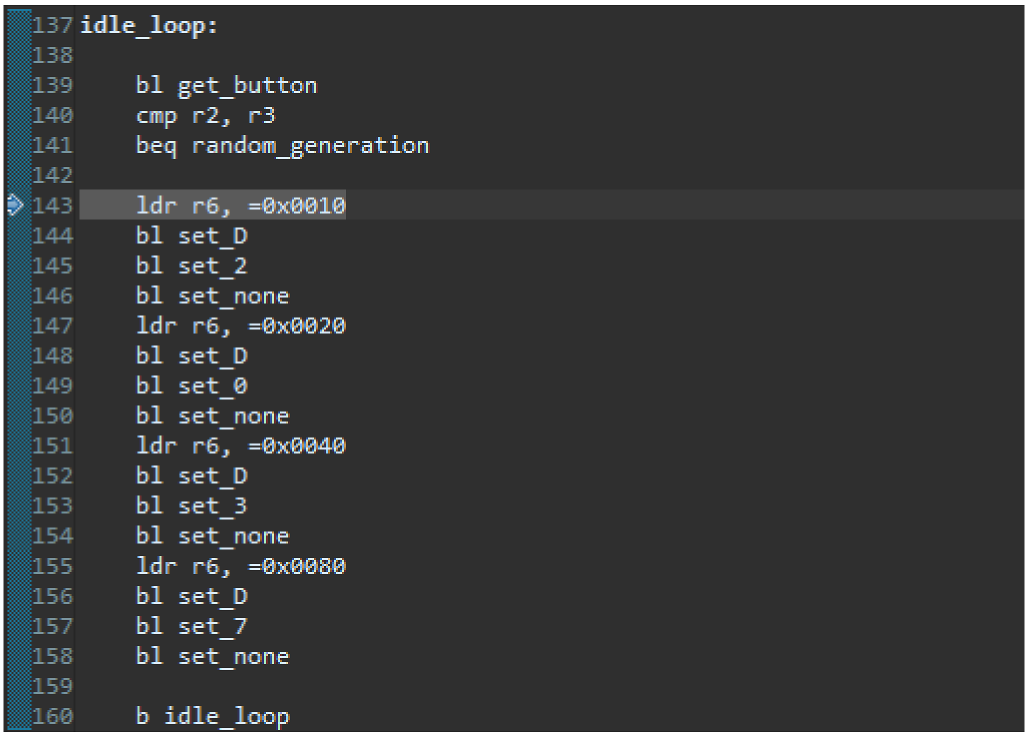
```
zero_delay:

    bl set_led_on

    ldr r7,=0x12AAA /* cycles for 1 sec total delay C271000 */
    bl second_delay
    b idle_loop

random_generation:

    pop {r0}
    bl generate_random
    push {r0}
    b countdown

countdown:

    bl number_extraction
    ldr r7,=0x26 /* cycles for 1 sec total delay C271000 */
    bl second_delay

    bl set_led_off

    bl get_button
    cmp r2, r3
    beq countdown_halt_loop

    subs r0, #0x1
    cmp r0, #0x0
    bne countdown
    b zero_delay

countdown_halt_loop:

    bl number_extraction
    ldr r7,=0x26 /* cycles for 1 sec total delay C271000 */
    bl second_delay

    bl get_button
    cmp r2, r3
    beq countdown

    b countdown_halt_loop
```

Main code uses all the other functions defined before it. Additionally it has new defined functions that create delays, refreshes, check for button press or turn the LED on and off.

These sub functions are needed to keep the code short enough so that we don’t get a offset error on pc. The code ahs additional comments that explain what parts do what.

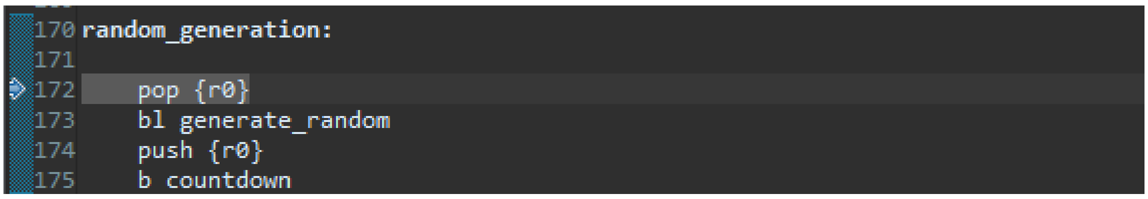
2.6 Testing of the Combined Code

The code will be tested on STM 32 IDE without any peripherals connected. After the code is sure to be working right we can connect all the peripherals properly and test the whole project.



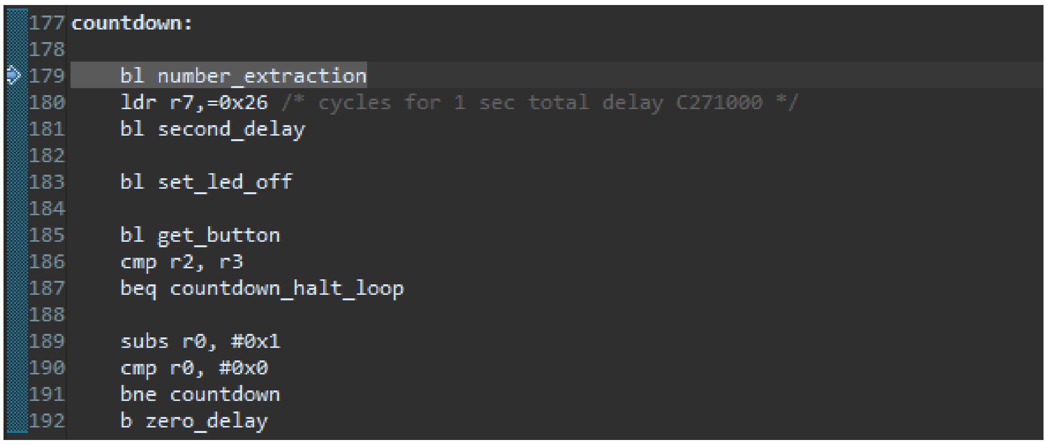
```
idle_loop:

    bl get_button
    cmp r2, r3
    beq random_generation

    ldr r6, =0x0010
    bl set_D
    bl set_2
    bl set_none
    ldr r6, =0x0020
    bl set_D
    bl set_0
    bl set_none
    ldr r6, =0x0040
    bl set_D
    bl set_3
    bl set_none
    ldr r6, =0x0080
    bl set_D
    bl set_7
    bl set_none

    b idle_loop
```

As we can see when the button is not pressed the it stays inside the idle loop where it shows the number 2037 which is the last 4 digits of my school number.



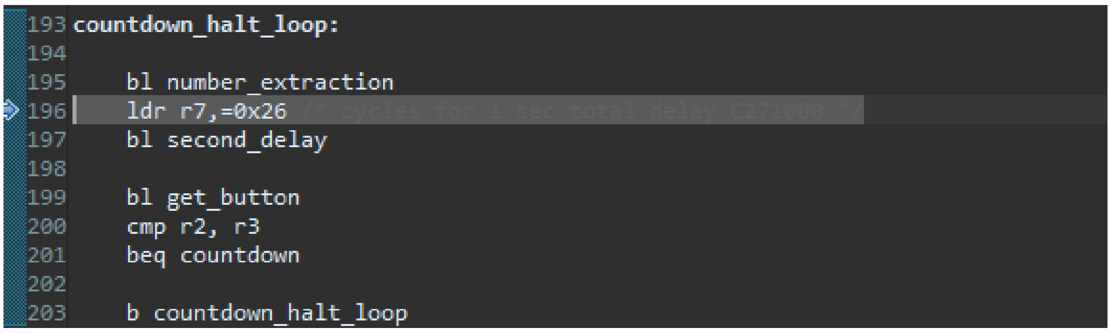
```
random_generation:

    pop {r0}
    bl generate_random
    push {r0}
    b countdown
```

When the button is pressed however it jumps to the random generation loop where it will create a random number and jump into the countdown loop.



```
countdown:

    bl number_extraction
    ldr r7,=0x26 /* cycles for 1 sec total delay C271000 */
    bl second_delay

    bl set_led_off

    bl get_button
    cmp r2, r3
    beq countdown_halt_loop

    subs r0, #0x1
    cmp r0, #0x0
    bne countdown
    b zero_delay
```

The jump to countdown is successful aswell and the number is counted down up to 0.

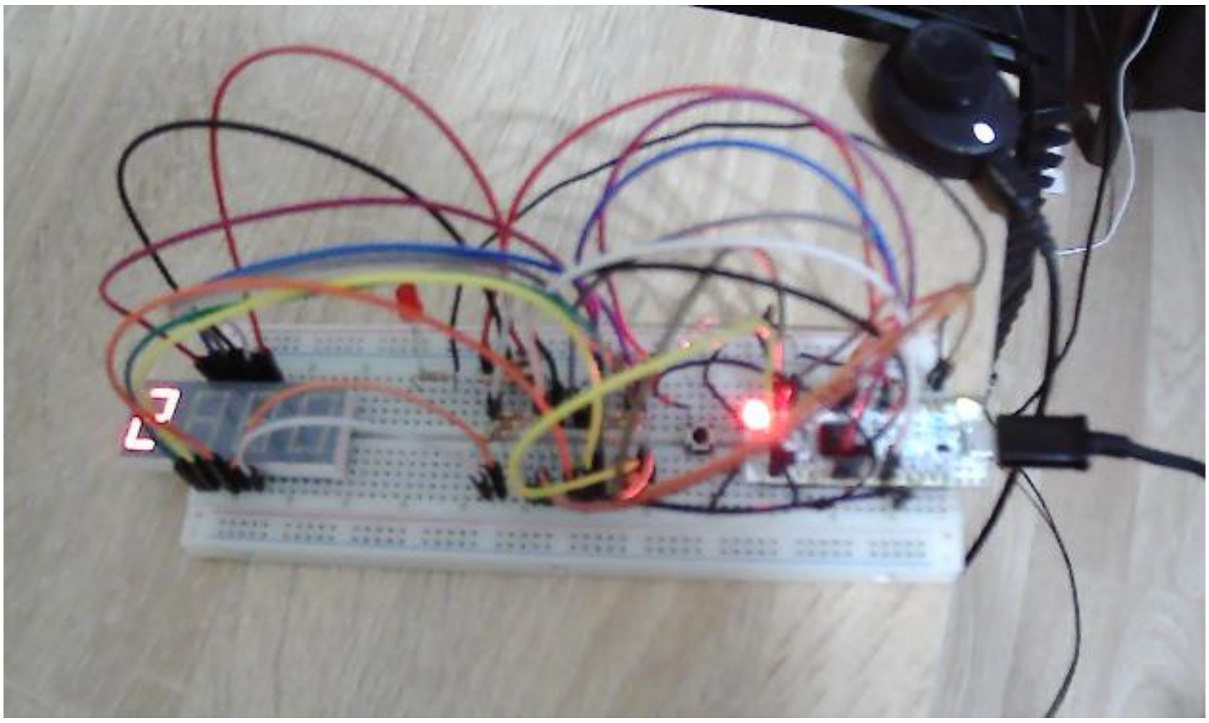
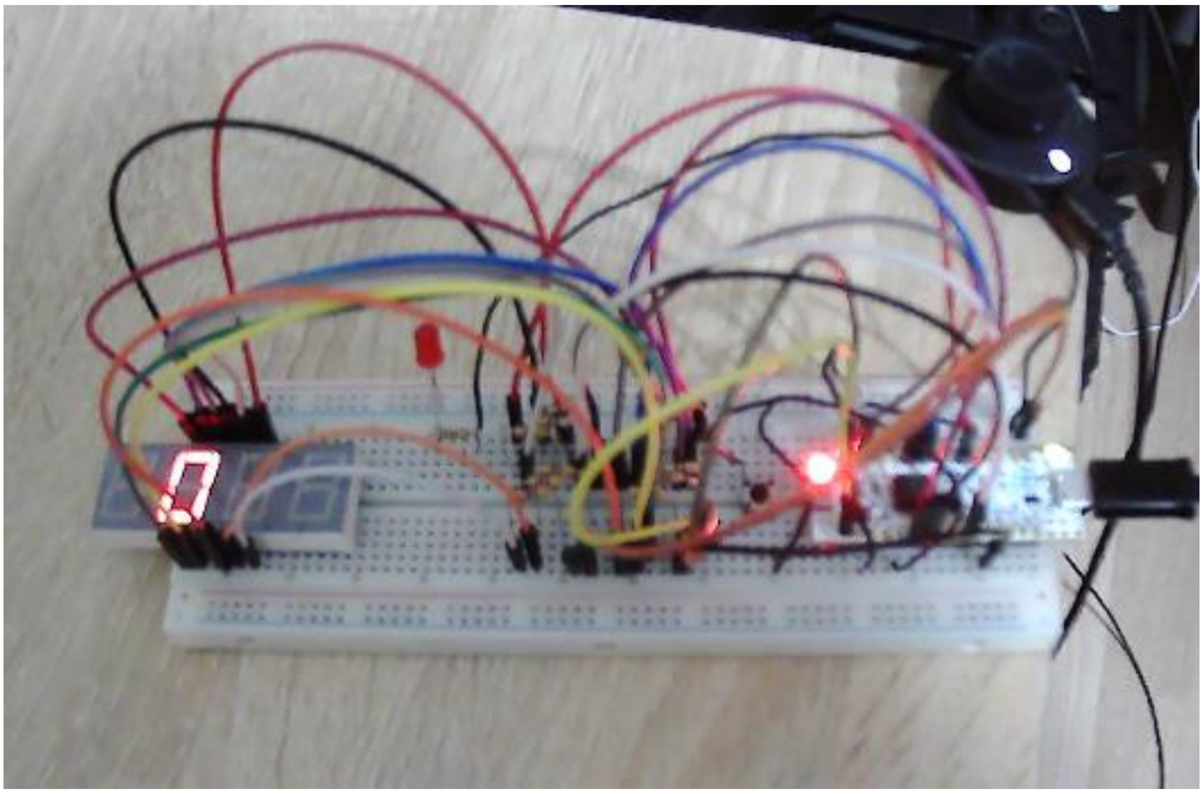


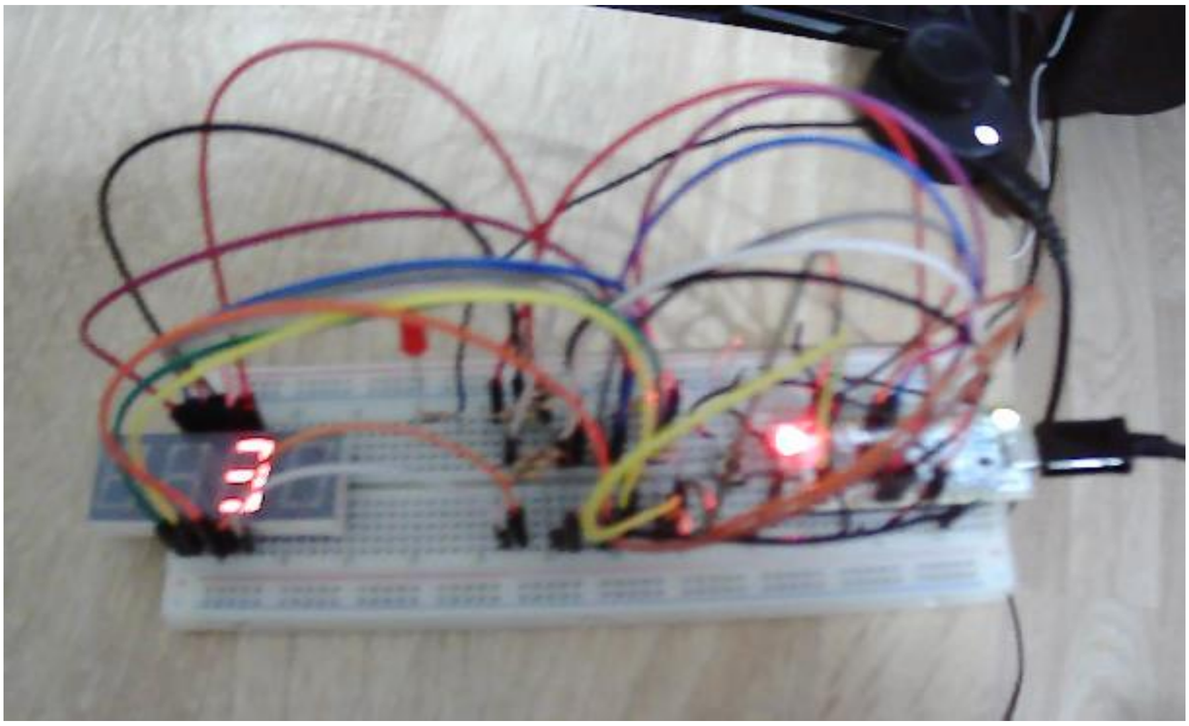
```
countdown_halt_loop:

    bl number_extraction
    ldr r7,=0x26 /* cycles for 1 sec total delay C271000 */
    bl second_delay

    bl get_button
    cmp r2, r3
    beq countdown

    b countdown_halt_loop
```

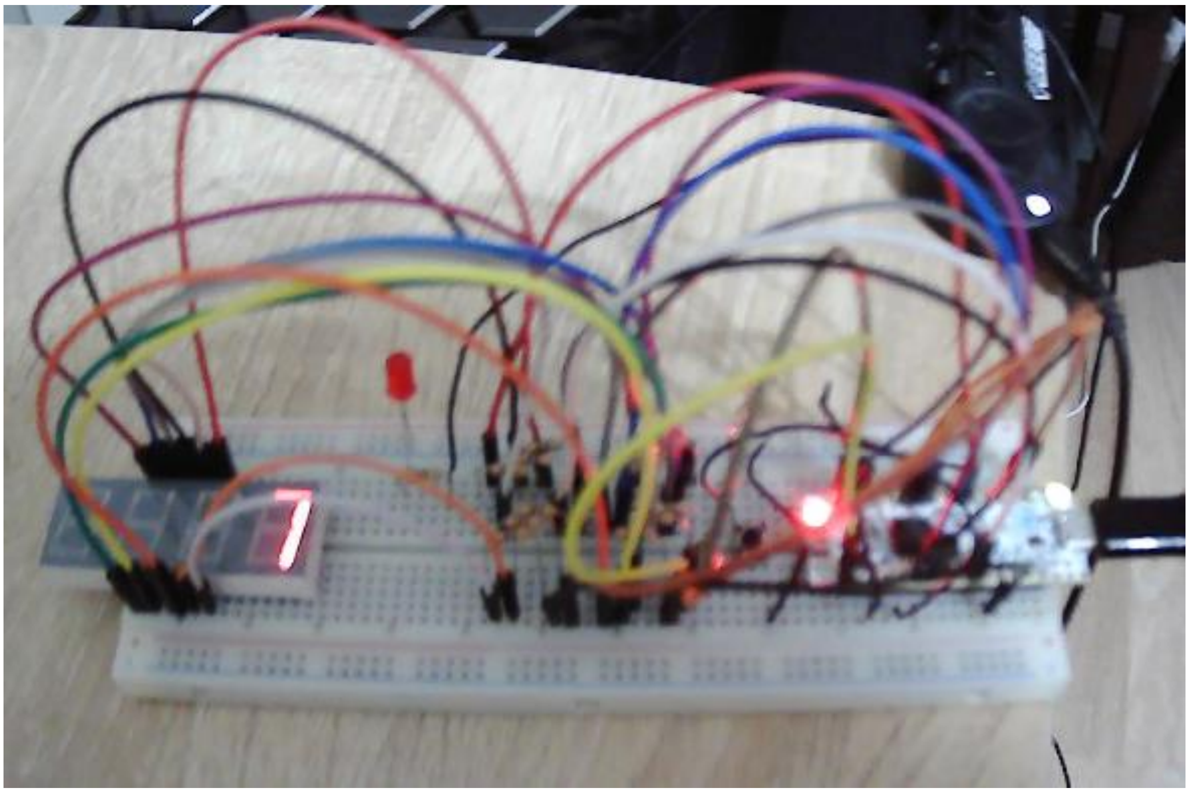
As we can see the countdown halt loop works aswell when the button is pressed in countdown.

2.7 Testing of the Board with Peripherals

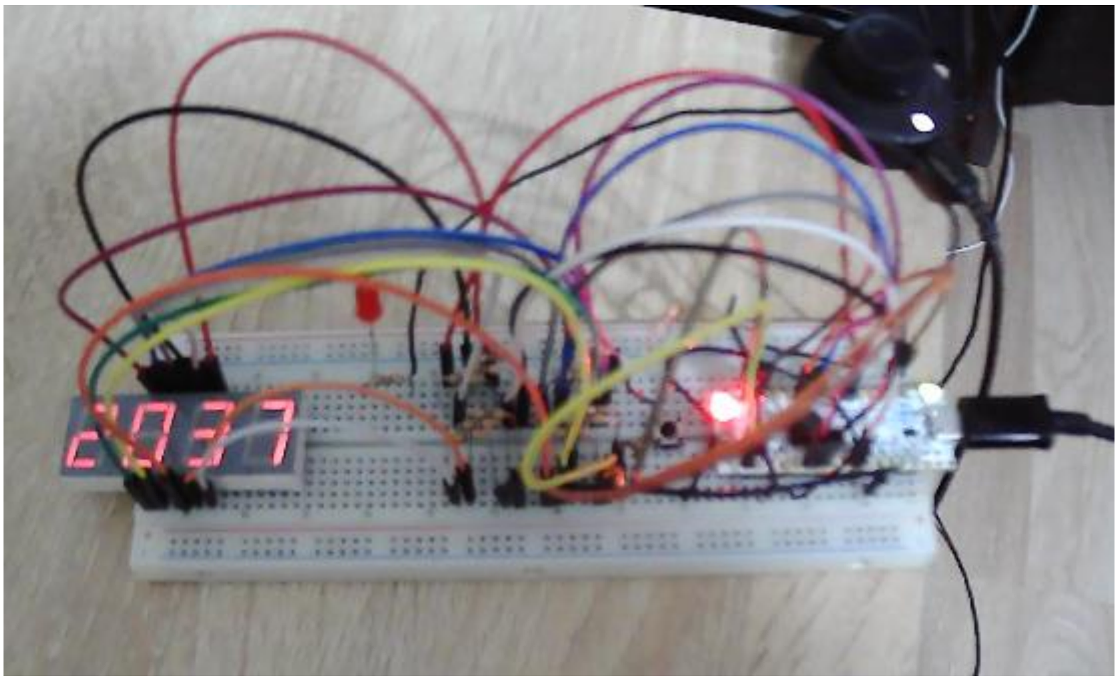
After the peripherals are connected we can test it on the peripherals. Code will be first run on debug mode and after deciding that it works correctly, we can test run it at full board speed.



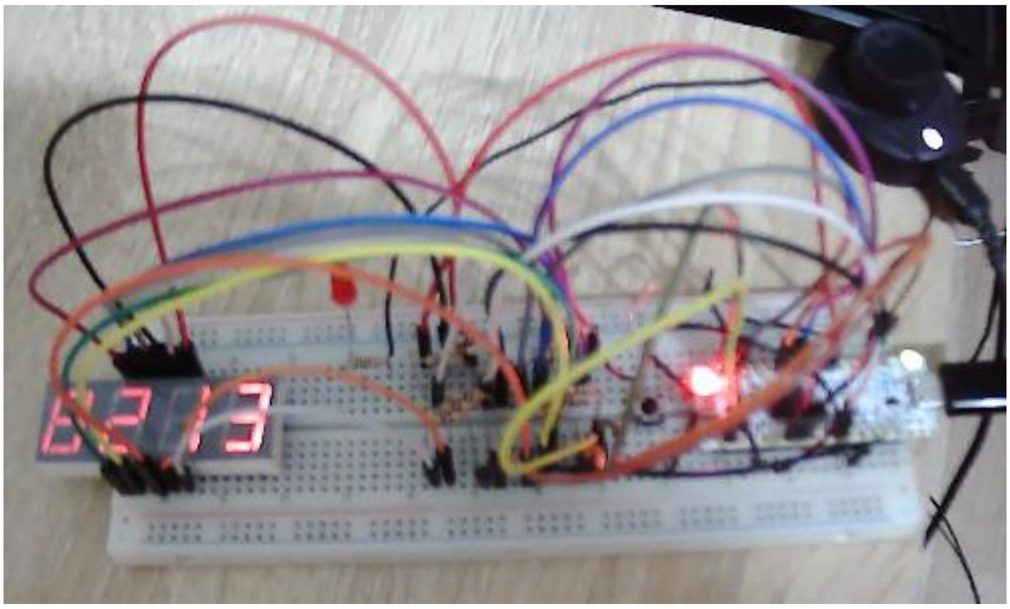


As we can see the board shows he correct digits in the correct pattern at the idle state. Now the test can be done on full speed run.

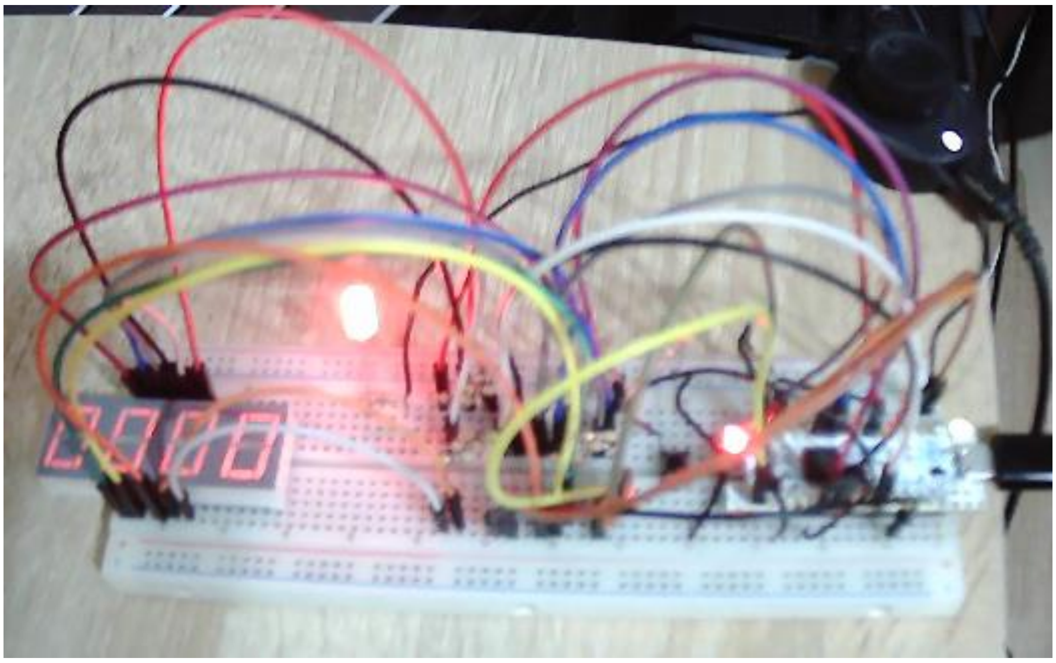


As we can see on the full speed run the idle state is shown to be working and showing the last 4 digits of my school number.

As for the random number generation the number generation works and shows the number when in countdown halt state.



Lastly when the countdown is over it shows 0 and the LED lights up to show that the countdown is over.



2.8 Prices and Parts List

Parts that are used in the project can be written into the table down below.

	Part Name	**Amount**	**Price**
1	Breadboard	1	7.50TL
2	Jumper Cable (Male-Male)	28	40 piece is around 3.16TL
3	LED (any colour preferably red)	1	0.16TL
4	470 ohm resistors	13	~
5	156 ohm resistor	1	~
6	Seven Segment Display	1	7.99TL
7	Button	1	0.27TL
8	STM32G031K8T6 Board	1	102.50TL
			TOTAL
			121.58TL

Total price list comes to around 121.58TL excluding the cargo costs. Build takes around 5 to 10 minutes using the block diagram given at the beginning of the report.

2.9 Video and Documenting the Project

Video documentation of the project will be loaded into the youtube. The link is given below.

https://www.youtube.com/watch?v=eFyX_fUBXrI

3. Results and General Comments

The results were as expected. I’ve learned to design a code implement it to a board and build a project on top of it all. The code had a lot of problems because of the program counter offset having a maximum value. This limited the length of the code dramatically and there were a lot of shortening of the code. But in the end the project works and functions as it should be.

As general comments there were a lot of debugging to see if thing were working properly or not. These debugging sessions could get confusing especially when the onboard written pin names are dramatically different than the software names of the pins. Lastly assembly language was hard to write the code on since it is a low level language.

4. References

[1]. RM0444 Reference manual
https://www.st.com/resource/en/reference_manual/dm00371828-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[2]. STM32G031K8 Datasheet

https://www.st.com/en/microcontrollers-microprocessors/stm32g031k8.html