GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC335 – MICROPROCESSORS LABAORATORY

LAB 2

HAZIRLAYANLAR
1801022035 – Ruveyda Dilara Günal
1801022071 – Alperen Arslan
1901022255 – Emirhan Köse

PROBLEMS

Problem 1

For this problem, you are asked to implement a diamond pattern given in Table 1 using external LEDs.

- Connect 8 LEDs and 1 push button to the board.
- The 8th LED should be a different color to indicate the status of the program. Let's call this **the status LED.**

Requirements:

- The button should be used to play or pause the pattern. You can assume the program has two modes: play, and pause, and the button is used to change modes.
- When in pause mode, the status LED should be on, and when in play mode, the status LED should be off.
- There should be around 125 ms delay between transitions. (i.e., t3-t2 ~= 125 ms)
- All the patterns are given in Table1. You should repeat these patterns indefinitely.

	LED1	LED2	LED3	LED4	LED5	LED6	LED7
t0							
t1				■			
t2			■	■	■		
t3		■	■	■	■	■	
t4	■	■	■	■	■	■	■
t5		■	■	■	■	■	
t6			■	■	■		
t7				■			

Table 1: Pattern on LEDs

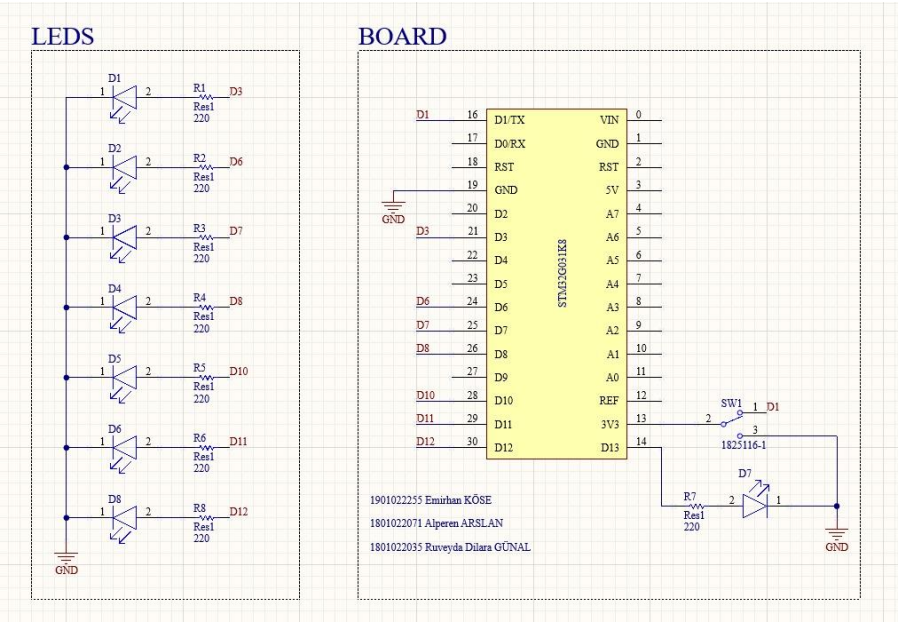

Figure 1: Problem 1 Block Diagram

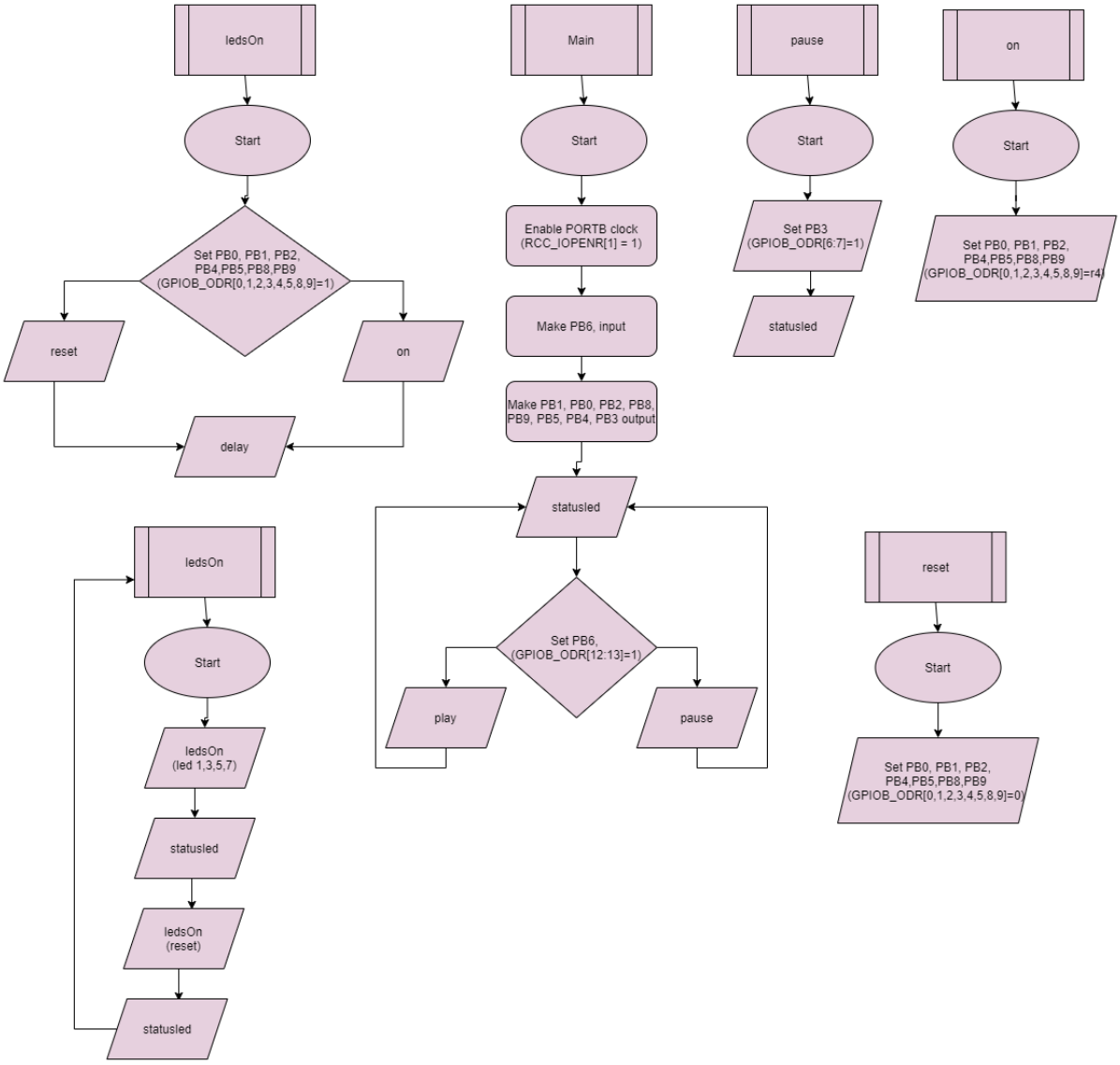

Figure 2:Problem 1 Flowchart

```
// Q1.s
 // Arrangement: Emirhan Köse, Ruveyda Dilara Günal, Alperen Arslan

.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 clock base and enable addresses
.equ RCC_BASE,         (0x40021000)          // RCC base address
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR register
offset

// define GPIO Base, Moder and ODR pin addresses
.equ GPIOB_BASE,       (0x50000400)          // GPIOB base address
.equ GPIOB_MODER,      (GPIOB_BASE + (0x00)) // GPIOB MODER
register offset
.equ GPIOB_IDR,        (GPIOB_BASE + (0x10)) // GPIOB IDR register
offset
.equ GPIOB_ODR,        (GPIOB_BASE + (0x14)) // GPIOB ODR register
offset

//Delay Interval
.equ delayInterval, 160000

// 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:
      // enable GPIOB clock, bit1 on IOPENR
      ldr r6, =RCC_IOPENR
      ldr r5, [r6]
      // movs expects imm8, so this should be fine
      movs r4, 0x2
      orrs r5, r5, r4
      str r5, [r6]

      // setup PB0, PB1, PB2...PB9 for 01 (Except PB7) and PB6 for
00 in MODER
```

```
      ldr r6, =GPIOB_MODER
      ldr r5, [r6]
      // cannot do with movs, so use pc relative
      ldr r5, =[0xFFFFF]
      str r5, [r6]
      ldr r4, =[0x5C555]
      ands r5, r5, r4
      str r5, [r6]

      bl statusLed //Control the status switch

      play:
      //First Stage
      ldr r4, =[0x100] //First leds connected to PB8
      bl ledsOn //Turn leds on
      bl statusLed //Control the status switch

      //Third Stage
      ldr r4, =[0x304] //Second and third leds connectod to PB2 and
PB9
      bl ledsOn
      bl statusLed

      //Fifth Stage
      ldr r4, =[0x325] //Fourth and Fifth leds connected to PB0 and
PB5
      bl ledsOn
      bl statusLed

      //Seventh Stage
      ldr r4, =[0x337] //Sixth and Seventh leds connected to PB1
and PB4
      bl ledsOn
      bl statusLed

      //Reset Stage
      ldr r4, =[0x000]
      bl ledsOn
      bl statusLed

      b play

      pause:
      ldr r6, = GPIOB_ODR
      ldr r5, [r6] //ODR Value
      movs r4, 0x8 //Status led connected to PB3
      orrs r5, r5, r4 //Setting led on
      str r5, [r6]
```

```
	b statusLed

	ledsOn:
	ldr r6, =GPIOB_ODR
	ldr r5, [r6]
	cmp r4,0x0 //Control the which led on at last
	beq Reset //If all leds are on, then take all them off
	bne On
	Reset:
	ands r5, r5, r4
	On:
	orrs r5, r5, r4
	str r5, [r6]
	// Assign value to register r1 to sub 1 per clock
	ldr r1, =delayInterval
	delay:
	subs r1, r1, #1
	bne delay
	bx lr

	statusLed:
	ldr r6, = GPIOB_IDR
	ldr r5, [r6] //IDR Value
	movs r4, #0x40 //Status switch connected to PB6
	ands r5, r5, r4 //Getting the value of button pressed or not
	lsrs r5, #6 //Shifting to lsb for compare

	cmp r5, #0x1 //Compare IDR Value with 1 bit
	bne BNE //If not equal
	beq BEQ //If equal

	//If is equal
	BEQ:
	b pause

	//If is not equal
	BNE:
	//Status Led Off
	ldr r6, =GPIOB_ODR
	ldr r5, [r6]
	ldr r5, =[0x0]
	str r5, [r6]
	bx lr

	// this should never get executed
	nop
```

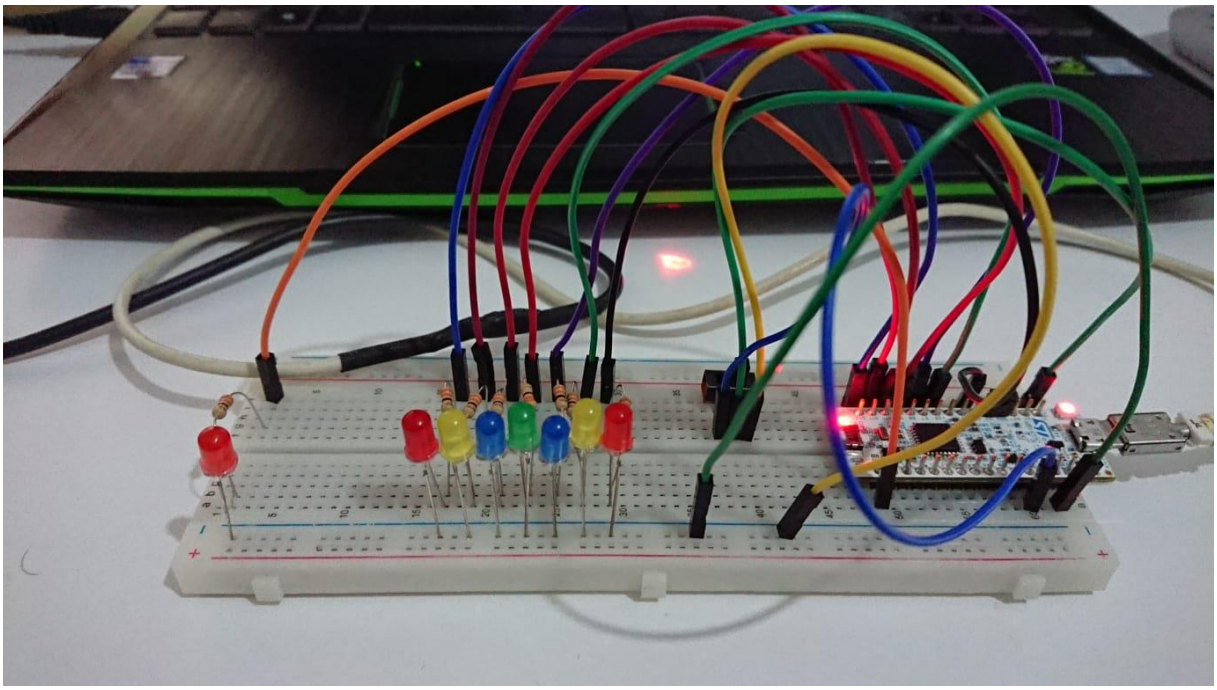

Figure 3: Leds at the t0

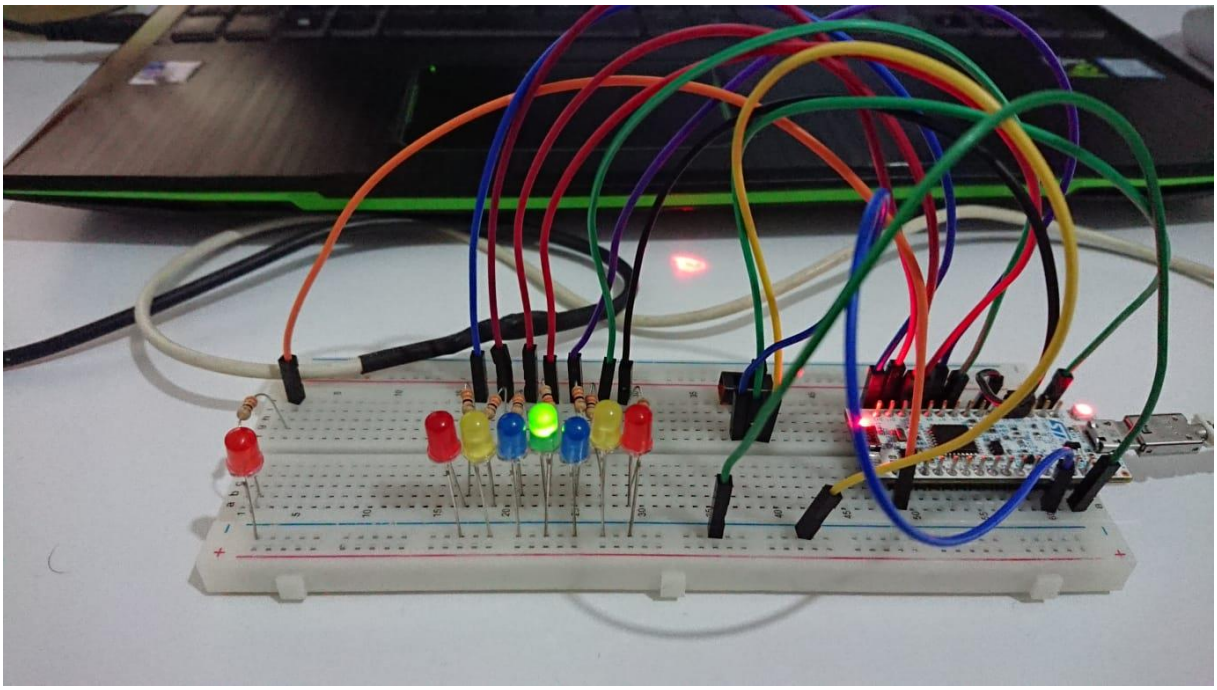

Figure 4: Leds at the t1 - t7

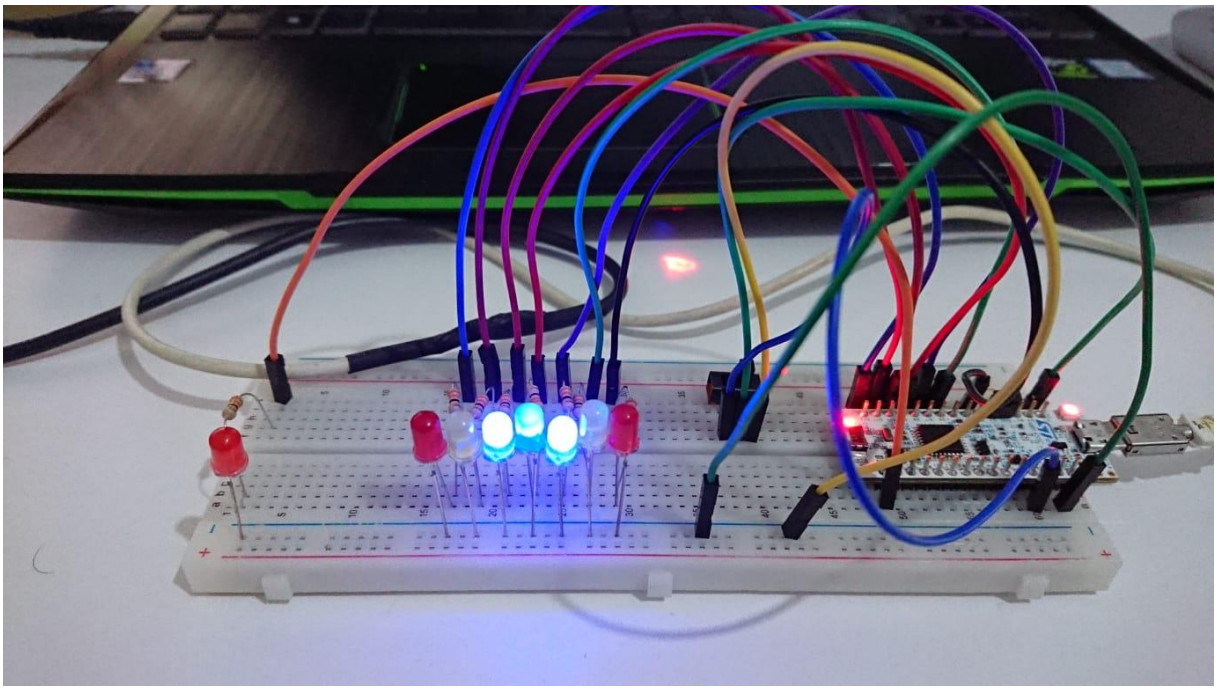

Figure 5: Leds at the t2 - t6

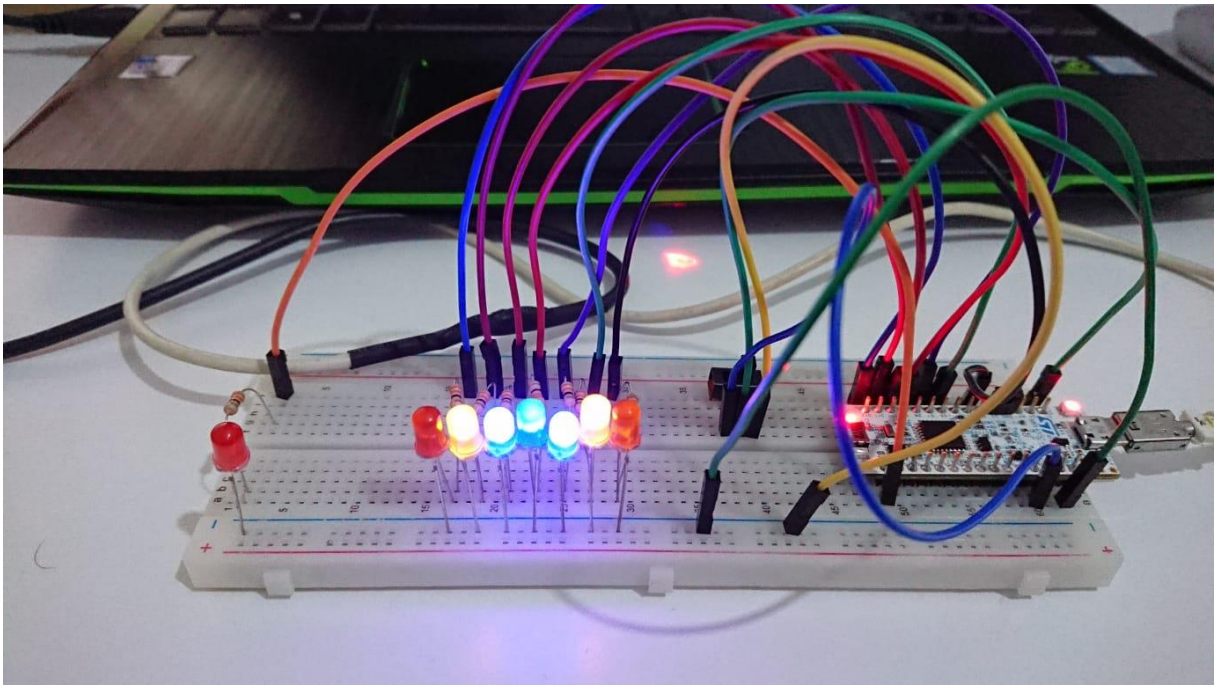

Figure 6: Leds at the t3 - t5

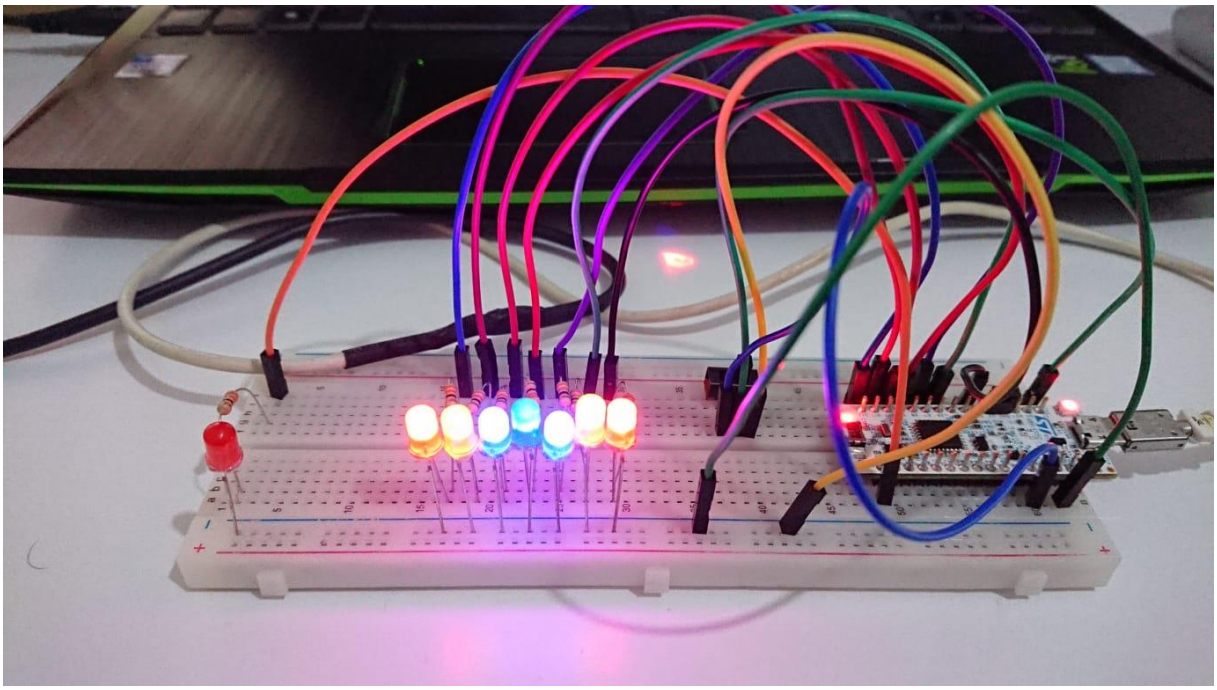

Figure 7: Leds at the t4

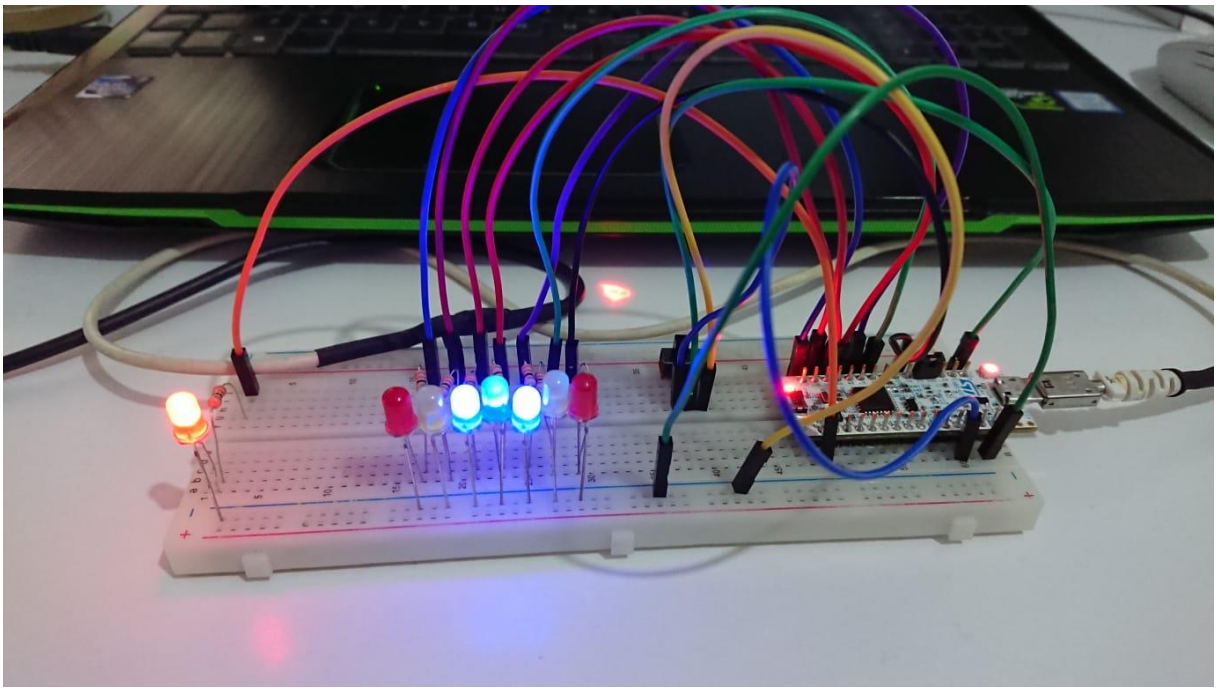

Figure 8: when the button is pressed, the LEDs

Questions:

● Using an oscilloscope, capture LED1, show the ON and OFF times. You can use a trigger mechanism to do this capture by setting it to one-time capture. ○ What happens if you decrease this delay time to 10 ms or less? Capture LED1 again and explain it.

● Capture both the button signal and the status LED. Then press the button for pause. ○ How long did it take to go from button press to status LED lighting up?

○ How about vice versa (press the button for play)?

○ Explain your findings.

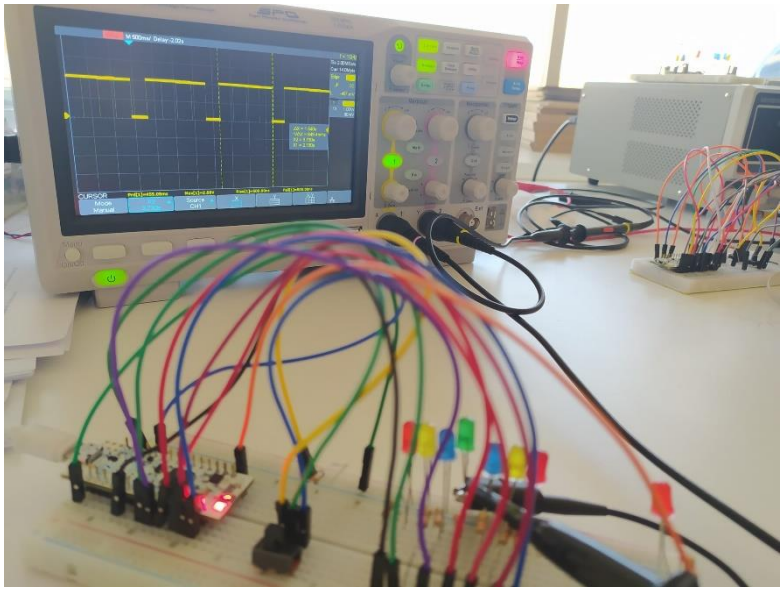

Figure 9:Voltage signal on the led

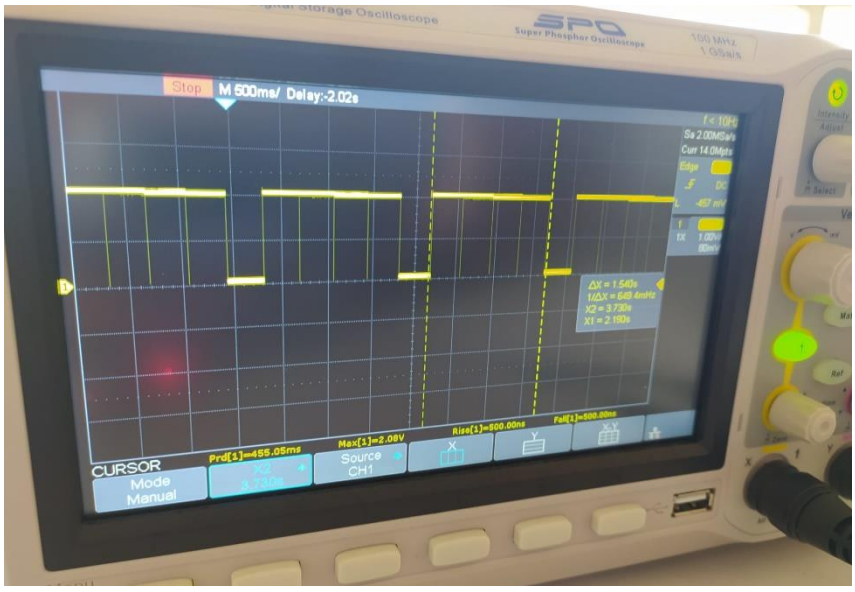

Figure 10:Voltage signal on the led

If we decrease the delay time to 10 ms it is going to be harder for us to observe the diamond pattern and the mode of LEDs.So here, we have decreased the on time of LEDs to 10 ms therefore, the diamond pattern started to distrupt. Moreover, making it less is going to cause of not seeing the diamond pattern in ongoing turns.

There is a latency of 2 ms when the button is pressed. When unpress the button there is latency of 6ms. It is the time of the signal that flows through the jumper. Latency can change. more or less. It is because of the environment we are measuring in is not ideal. It also can be depend on the operation that processor is doing at the time button pressed

Problem 2

In this problem you are asked to write a **decimal counter** using Seven Segment Displays.
● Connect 1 x Seven Segment Displays, 2 x buttons, and 1 status LED.

Requirements:

● SSD should display the last digit of your school ID.
● One button should cycle through each project member's ID on the SSD.
● Second button should start the automatic counting down from that number down to 0.
○ It should roughly go down at 1 second intervals. If the ID is 0, treat it as 10 and count down from 10.

● Upon reaching 0, it should stay there and wait there for any button press. ○ If the cycle button is pressed, it should display the next ID.
○ If the counter button is pressed, it should count down from the original ID again.

● Status LED should be ON when the countdown operation is in progress. OFF otherwise.

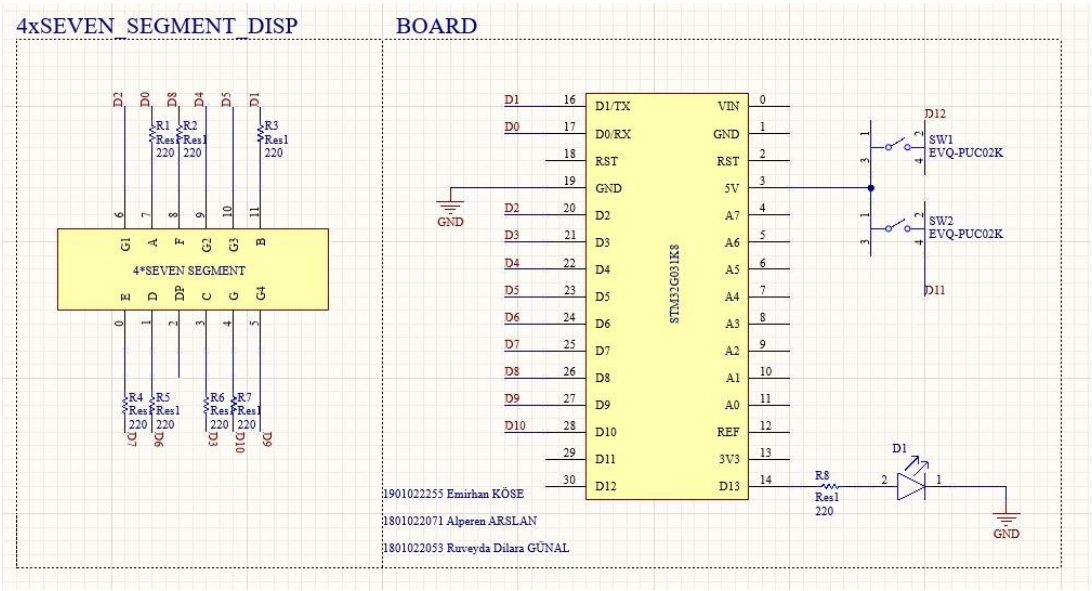

Figure 11: Problem 2 Block Diagram

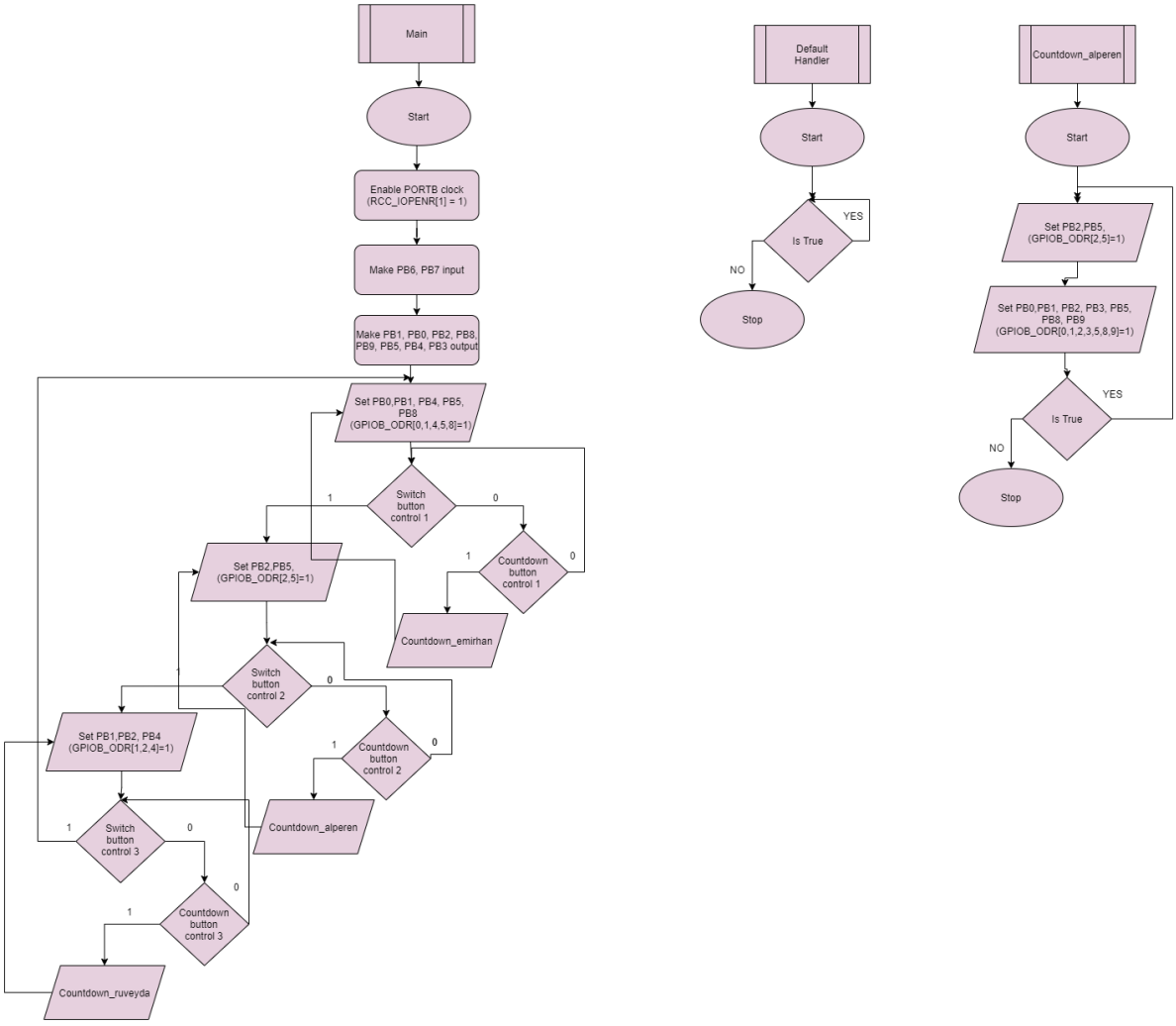

Figure 12: Problem 2 Flowchart

```
/*
 * asm.s
 *
 * authors: Emirhan KÖSE, Ruveyda Dilara GÜNAL, Alperen ARSLAN
 *
 * description: Added the necessary stuff for turning on the green
LED on the
 *   G031K8 Nucleo board. Mostly for teaching.
 */
.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 GPIOB_BASE,       (0x50000400)          // GPIOC base address
.equ GPIOB_MODER,      (GPIOB_BASE + (0x00)) // GPIOC MODER
register offset
.equ GPIOB_ODR,        (GPIOB_BASE + (0x14)) // GPIOC ODR register
offset
.equ GPIOB_IDR,    (GPIOB_BASE +(0x10))

/* 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:
/* PORT B is enabled*/
      ldr r6, =RCC_IOPENR
      ldr r5, [r6]
      ldr r4,=#2
      orrs r5,r5,r4
      str r5,[r6]

      ldr r6,=GPIOB_MODER
      ldr r5,[r6]
```

```
	ldr r4,=0xFFFFF
	mvns r4,r4
	ands r5,r5,r4
	ldr r4,=0x50555
	orrs r5,r5,r4
	str r5,[r6]

	emirhan:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x135
	orrs r5,r5,r4
	str r5,[r6]
	b switch_button_control_1

	alperen:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x22
	orrs r5,r5,r4
	str r5,[r6]
	b switch_button_control_2

	ruveyda:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x26
	orrs r5,r5,r4
	str r5,[r6]
	b switch_button_control_3

	switch_button_control_1:
	ldr r6,=GPIOB_IDR
	ldr r5,[r6]
	ldr r4,=0x175
	ldr r7,=0xf4240
	bl delay
	cmp r5,r4
	beq alperen
	bne countdown_button_1

	switch_button_control_2:
	ldr r6,=GPIOB_IDR
```

```
	ldr r5,[r6]
	ldr r4,=0x62
	ldr r7,=0xf4240
	bl delay
	cmp r5,r4
	beq ruveyda
	bne countdown_button_2

	switch_button_control_3:
	ldr r6,=GPIOB_IDR
	ldr r5,[r6]
	ldr r4,=0x66
	ldr r7,=0xf4240
	bl delay
	cmp r5,r4
	beq emirhan
	bne countdown_button_3

	countdown_button_1:
	ldr r6,=GPIOB_IDR
	ldr r5,[r6]
	ldr r4,=0x1B5
	ldr r7,=0xf4240
	bl delay
	cmp r5,r4
	beq countdown_emirhan
	bne switch_button_control_1

	countdown_button_2:
	ldr r6,=GPIOB_IDR
	ldr r5,[r6]
	ldr r4,=0xA2
	ldr r7,=0xf4240
	bl delay
	cmp r5,r4
	beq countdown_alperen
	bne switch_button_control_2

	countdown_button_3:
	ldr r6,=GPIOB_IDR
	ldr r5,[r6]
	ldr r4,=0xA6
	ldr r7,=0xf4240
	bl delay
	cmp r5,r4
	beq countdown_ruveyda
	bne switch_button_control_3
	countdown_emirhan:
	bl number_5
```

```
	ldr r7,=0xf4240
	bl delay
	bl number_4
	ldr r7,=0xf4240
	bl delay
	bl number_3
	ldr r7,=0xf4240
	bl delay
	bl number_2
	ldr r7,=0xf4240
	bl delay
	bl number_1
	ldr r7,=0xf4240
	bl delay
	bl number_0
	ldr r7,=0xf4240
	bl delay
	b emirhan

	countdown_alperen:
	bl number_1
	ldr r7,=0xf4240
	bl delay
	bl number_0
	ldr r7,=0xf4240
	bl delay
	b alperen

	countdown_ruveyda:
	bl number_7
	ldr r7,=0xf4240
	bl delay
	bl number_6
	ldr r7,=0xf4240
	bl delay
	bl number_5
	ldr r7,=0xf4240
	bl delay
	bl number_4
	ldr r7,=0xf4240
	bl delay
	bl number_3
	ldr r7,=0xf4240
	bl delay
	bl number_2
	ldr r7,=0xf4240
	bl delay
	bl number_1
	ldr r7,=0xf4240
```

```
	bl delay
	bl number_0
	ldr r7,=0xf4240
	bl delay
	b ruveyda

	number_7:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x2E
	orrs r5,r5,r4
	str r5,[r6]
	bx lr

	number_6:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x33D
	orrs r5,r5,r4
	str r5,[r6]
	bx lr

	number_5:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x13D
	orrs r5,r5,r4
	str r5,[r6]
	bx lr

	number_4:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x3B
	orrs r5,r5,r4
	str r5,[r6]
	bx lr

	number_3:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
```

```
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x13E
	orrs r5,r5,r4
	str r5,[r6]
	bx lr

	number_2:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x31E
	orrs r5,r5,r4
	str r5,[r6]
	bx lr

	number_1:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x2A
	orrs r5,r5,r4
	str r5,[r6]
	bx lr

	number_0:
	ldr r6,=GPIOB_ODR
	ldr r5,[r6]
	ldr r4,=0x0
	ands r5,r5,r4
	ldr r4,=0x32F
	orrs r5,r5,r4
	str r5,[r6]
	bx lr

	delay:
	subs r7,#1
	cmp r7,0x0
	bne delay
	bx lr

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

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

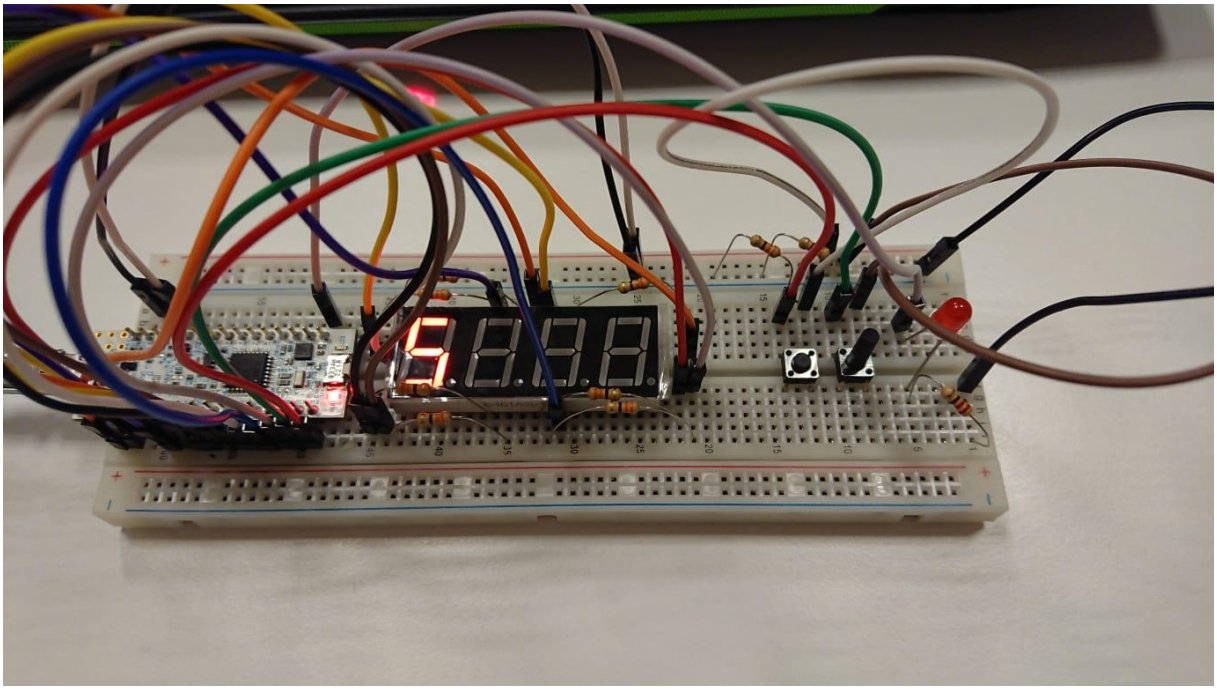

Figure 13: Display of the number to be counted down

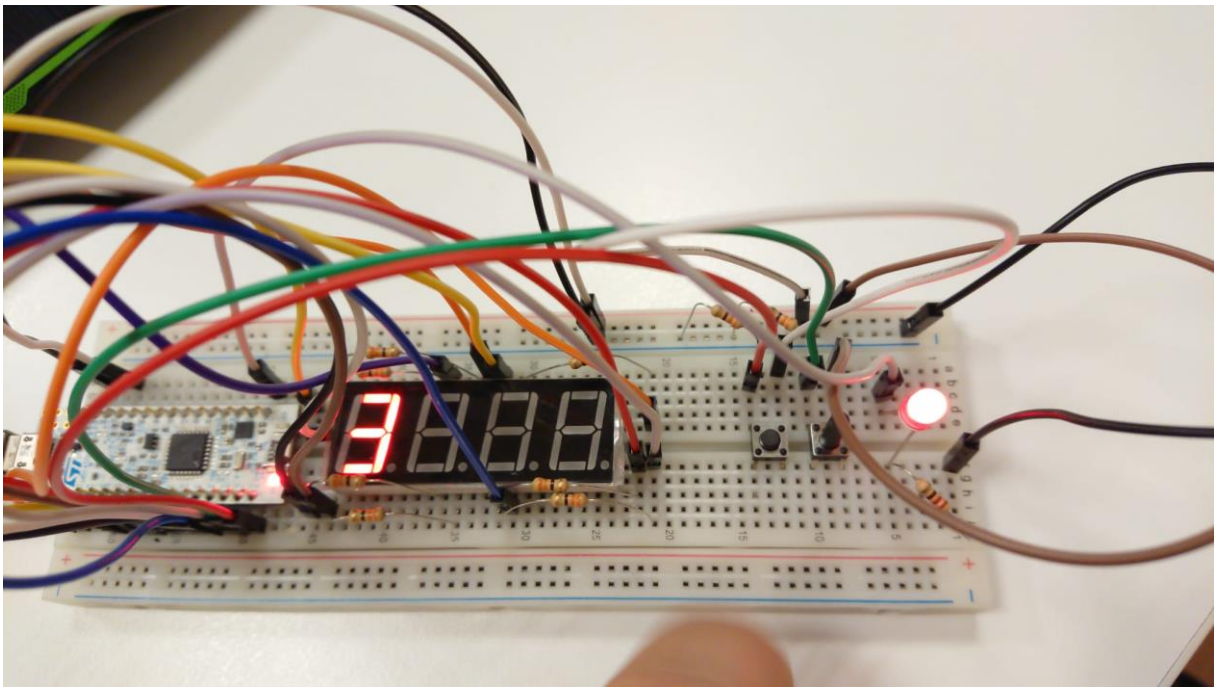

Figure 14: When counting down

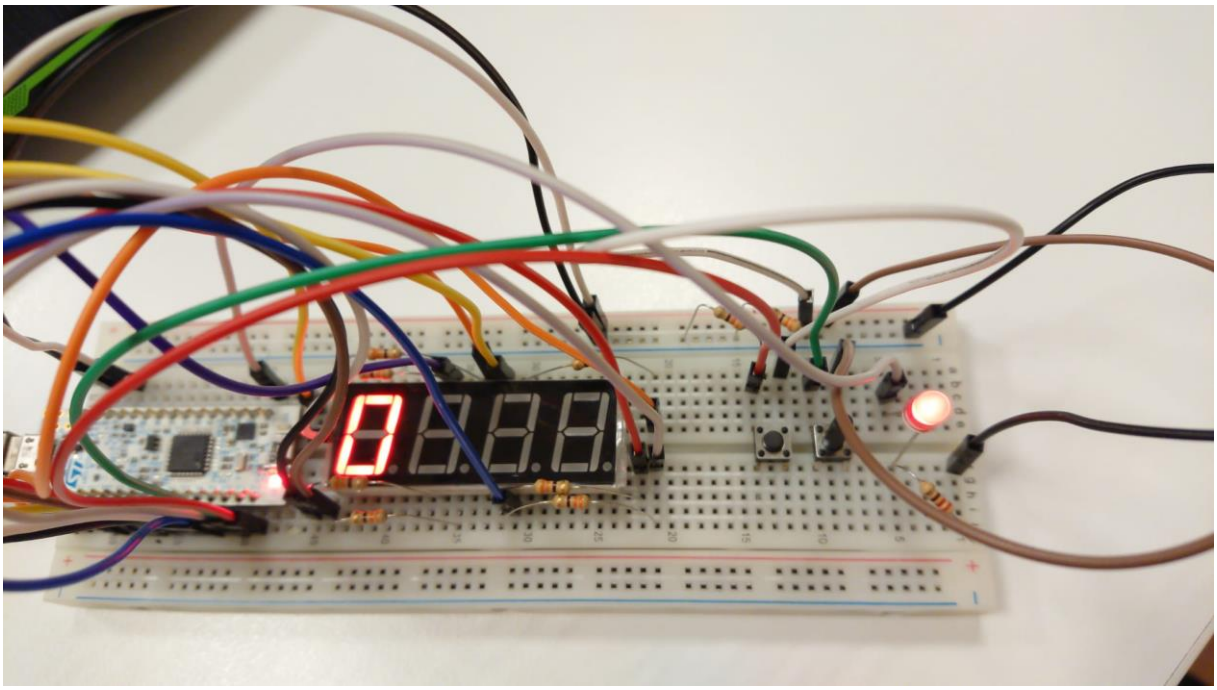

Figure 13: When pausing

Questions:

- Using an oscilloscope, capture the status LED when the countdown operation is in progress, and show the ON time. Does it match the seconds in the requirements?
- Do all the buttons need debouncing? Explain the method you implemented.

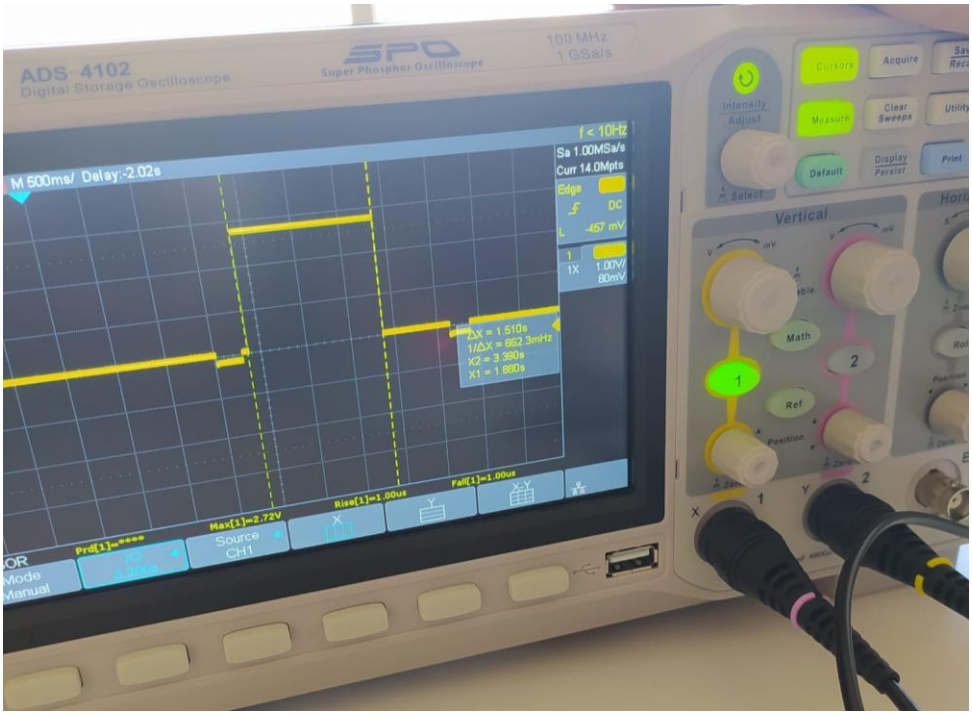

Figure 14: Voltage signal on the led

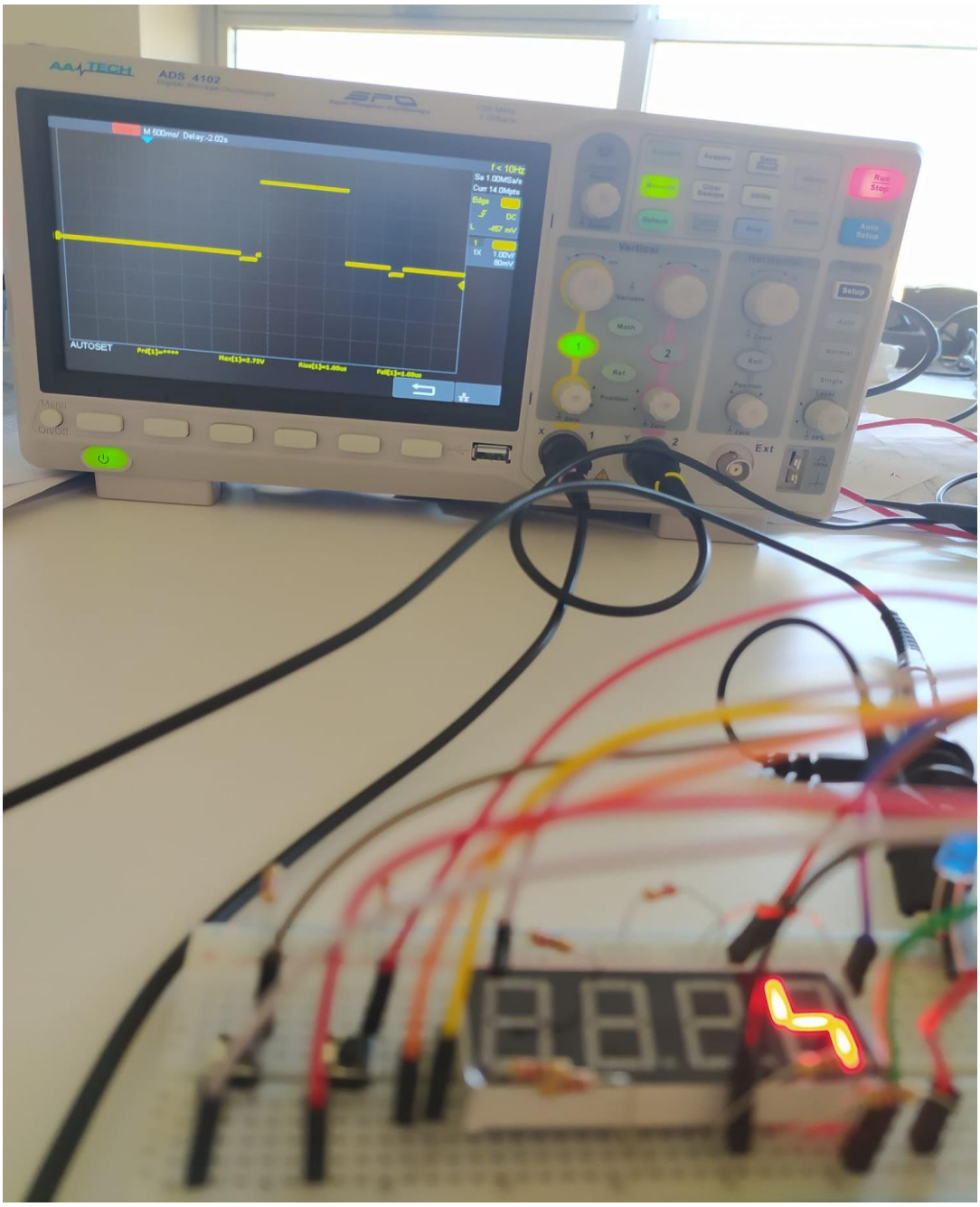

Figure 15: Voltage signal on the led

It took 1.5 seconds while status led on. And it doesn't provide the requirements. As in the picture cause we didn't notice in the question the time interval. All buttons don't need debouncing. Just switching button needs debouncing.

Conclusion

First of all, 7 leds were arranged side by side and the 8th led was added as a status led. These LEDs will flash in diamond shape at 125 millisecond intervals. Then, the voltage of the

middle led was measured with the help of an oscilloscope. Later, this time was brought to around 10 milliseconds and the voltage of the 1st led was measured with the help of an oscilloscope, and these values were added to the report as a picture. Then we wrote a program that counts the last digits of our school numbers backwards with a seven segment display. There are two buttons in this program. The first button changes the student numbers and the second button does the countdown. During this countdown process, the voltage of the status led was measured with an oscilloscope and the results were added to the report.7

Links to videos

https://youtu.be/zJCtuYku8L8

https://youtu.be/y2jtayTAJ1M

https://youtu.be/hnchKrb03qk

https://youtu.be/eVn_GBreHo4

https://youtu.be/uG1XgLGhEcU

Reference

https://github.com/fcayci/stm32g0

https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group1/05/c3/27/2a/6b/db/41/f1/MB1455-G031K8-C01_Schematic/files/MB1455-G031K8-C01_Schematic.pdf/jcr:content/translations/en.MB1455-G031K8-C01_Schematic.pdf

https://www.st.com/resource/en/reference_manual/rm0444-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf