Gebze Technical University
Electronics Engineering – Fall 2021

ELEC335 | Microprocessors Lab

LAB#2

Due Date	10.11.2021	
Student 1		
Student 2		
Student 3		

QUESTION 1

1.i) Description

EXPERIMENT CIRCUIT:

In the experiment, 8 leds are connected as OUTPUT, 7 leds are operational leds, 1 led is status led. Also a button is connected as INPUT for controlling Play and Pause modes.

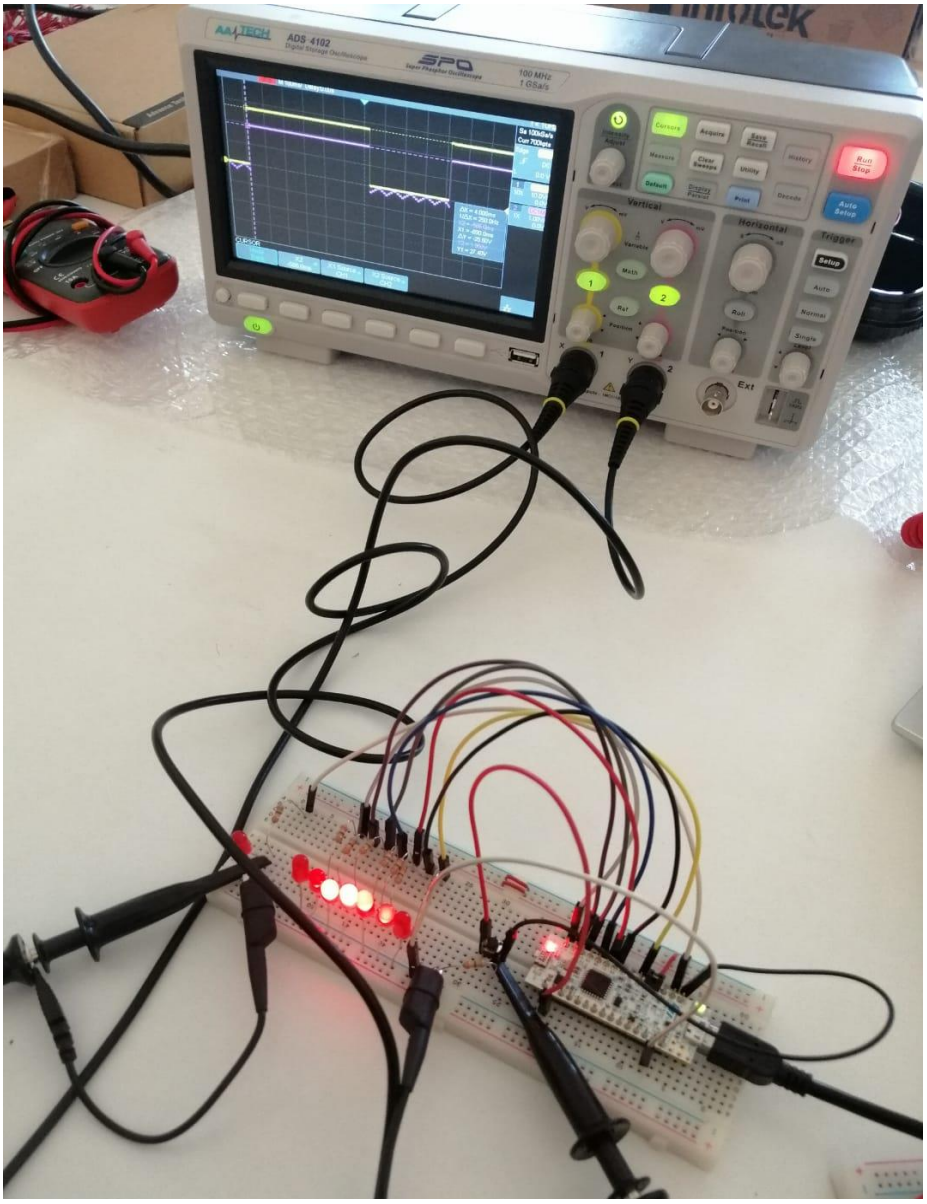


Figure 1: Experiment Circuit

PLAY MODE:

In play mode the status led is off and other 7 leds are playing.

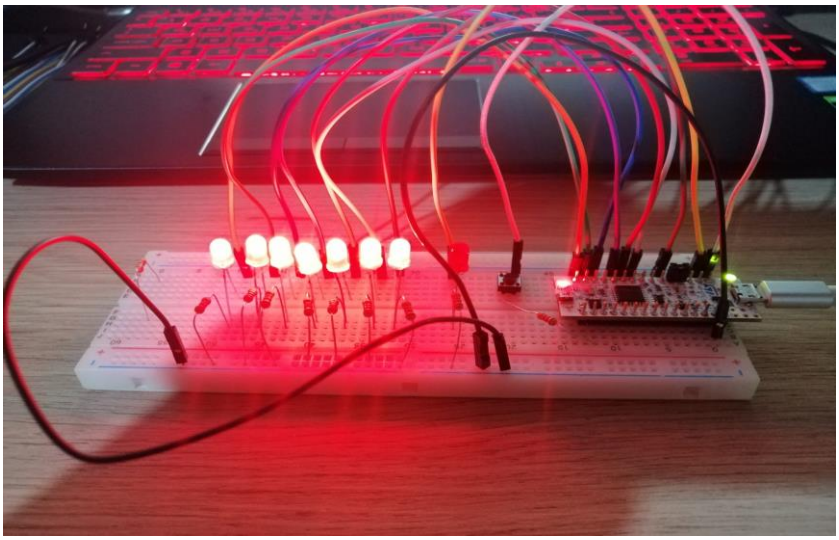


Figure 2: Play Mode, status led off

PAUSE MODE:

In pause mode the status led is on and other 7 leds are stop in what they are.

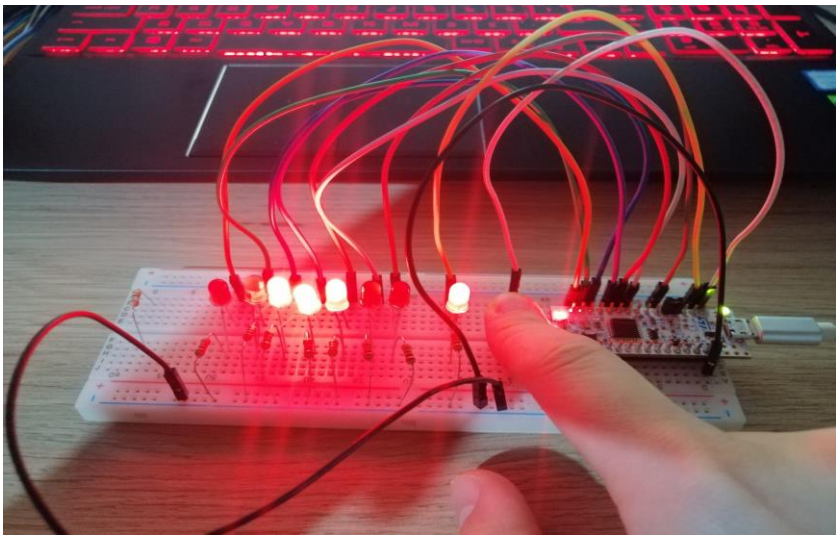


Figure 3: Pause Mode, status led on

Project Video: https://drive.google.com/file/d/1FULnO0pI2e9-DCRsJRxMrRrw9v0In1XZ/view?usp=sharing

1.ii) Assembly Code

```
/*
 * lab2_prob1.s
 *
 *
 * Description: turning on and off LEDs on the G031K8 Nucleo
board as play and pause mode in direction.
 */


.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)) // GPIOC
IDR 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 */
```

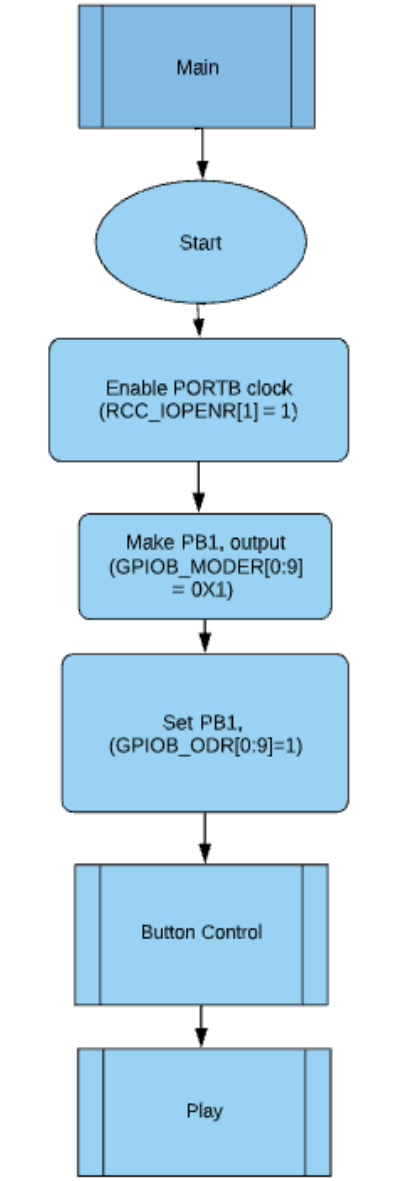
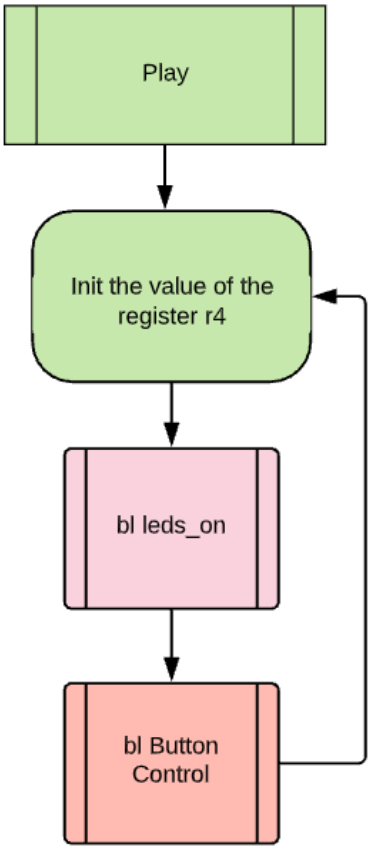
```
     /* add rest of them here if needed */


/* 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 */
/* main function */
.section .text
main:

     ldr r6,=RCC_IOPENR
     ldr r5,[r6]
     movs r4,#2 // activeted B port
     orrs r5,r5,r4
     str r5,[r6] //

     ldr r6,=GPIOB_MODER // in out mod
     ldr r5,[r6]
     ldr r4,=0xFFF3F // pins to use according to MODER
     mvns r4,r4
     ands r5,r5,r4
     ldr r4,= 0x54555
     orrs r5,r5,r4
     str r5,[r6]

     bl button_control


play:

     ldr r4,=0x100 // PB8 led in the middle t1
     bl leds_on
     bl button_control


     ldr r4,=0x304 // PB2 and PB9 t2
     bl leds_on
     bl button_control


     ldr r4,=0x325 //PB0 and PB5 t3
```

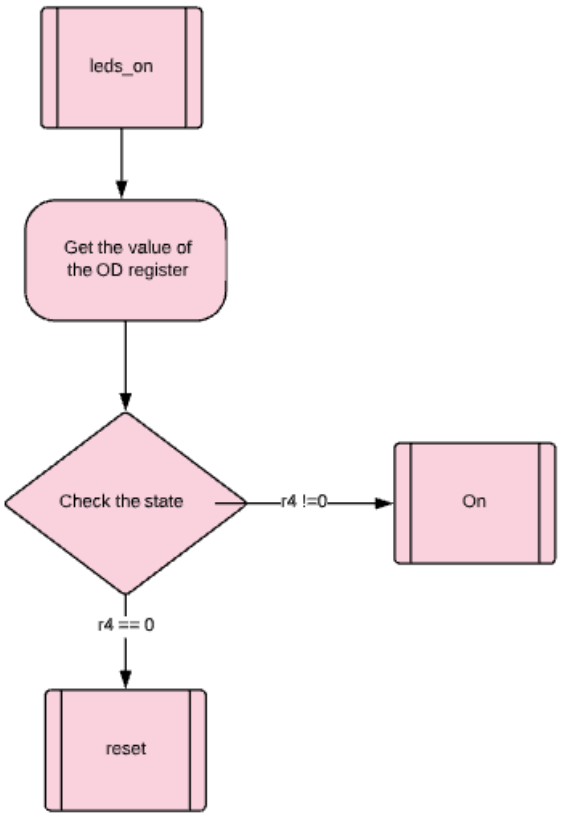
```
     bl leds_on
     bl button_control


     ldr r4,=0x337 //PB1 and PB4 t4
     bl leds_on
     bl button_control


     ldr r4,=0x325 // PB0 and PB5 t5
     bl leds_on
     bl button_control


     ldr r4,=0x304 // PB2 and PB9 t6
     bl leds_on
     bl button_control


     ldr r4,=0x100 //PB8 t7
     bl leds_on
     bl button_control


     b play

pause:
     ldr r6, = GPIOB_ODR
     ldr r5, [r6] //ODR Value
     ldr r4,=0x80 //Status led connected to PB7
     orrs r5, r5, r4 //Setting led on
     str r5, [r6]
     b button_control

     leds_on:
     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 r7 to sub 1 per clock
     ldr r7, =0x1E8480 // 125ms to hexadecimal
     delay:
     subs r7, r7, #1
```

```
     bne delay
     bx lr

     button_control:
     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

     BEQ:
     b pause

     BNE:
     //Status Led Off
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r5, =[0x0]
     str r5, [r6]
     bx lr


     nop
```

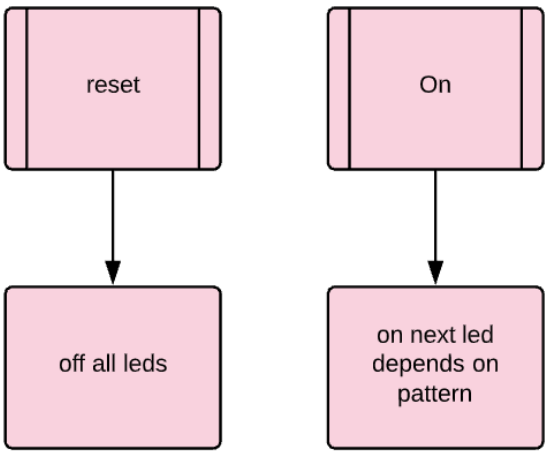
1.iii) Flowchart

1- Main Function	**2-** Play Function
	

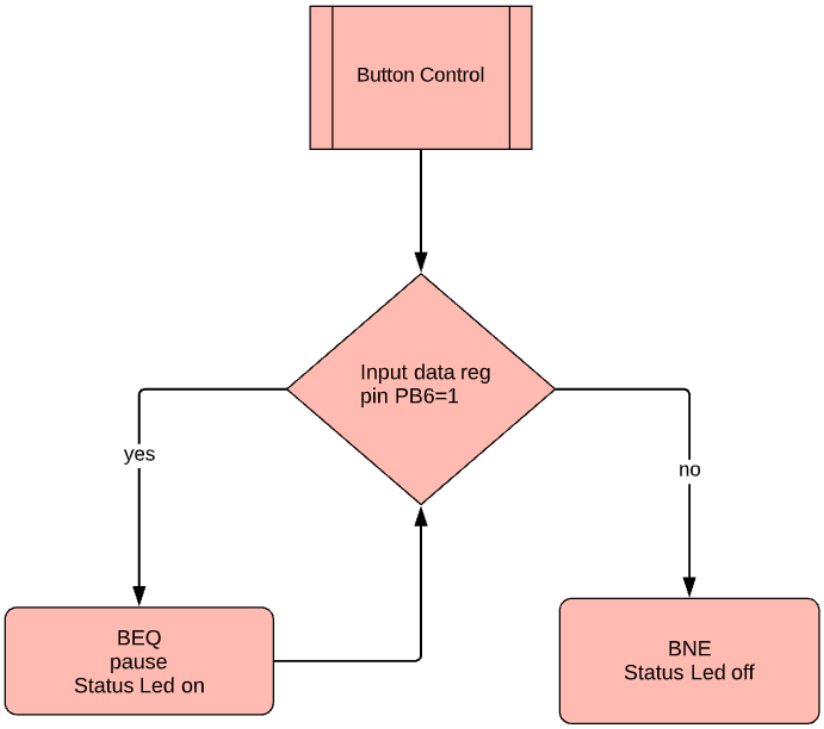
3- leds_on function



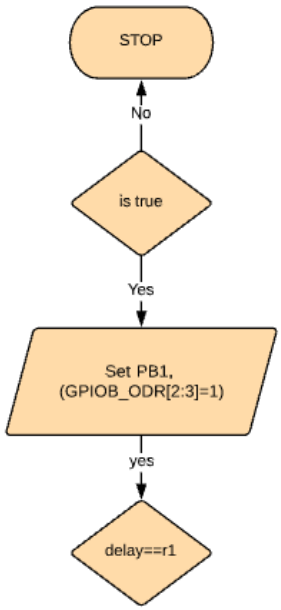
4- reset and on led functions



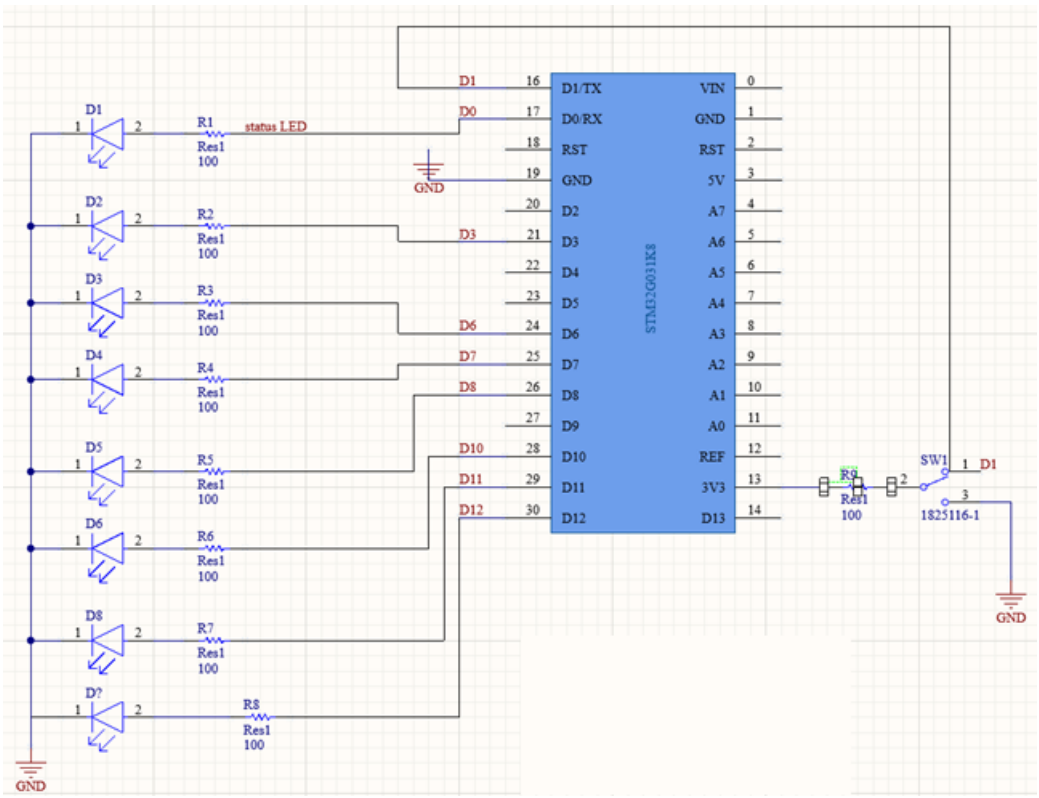
5- Button Control Function



6- Stop Function



1.iv) Circuit Scheme



1.v) Oscilloscope Measurements

- Prob connected to LED1 as in Figure 4, measured on and off times as in Figure 5,6 and 7. LED1 on 370 ms, off 2.200s and all period measured as 2.620 s.

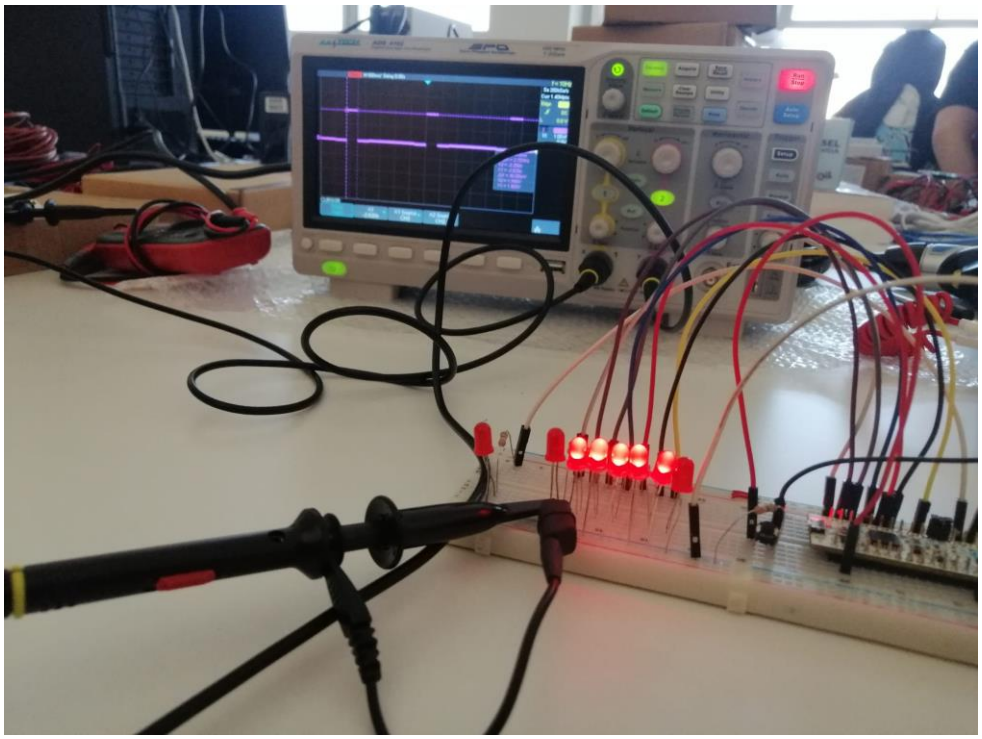


Figure 4: LED1, connected to oscilloscope

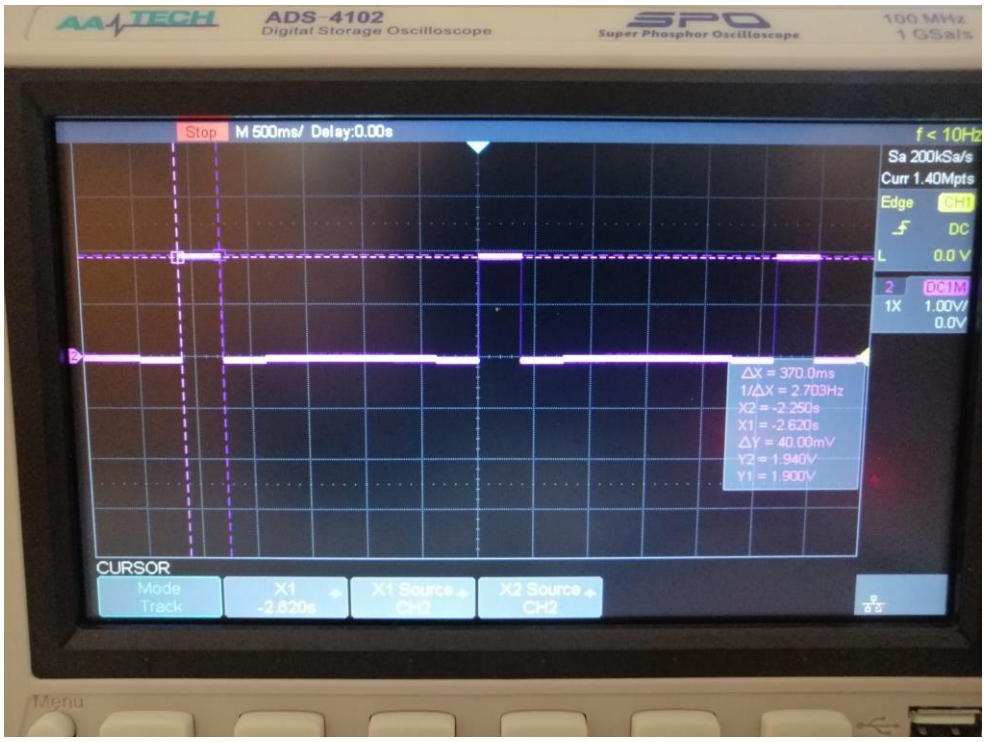


Figure 5: LED1, on time

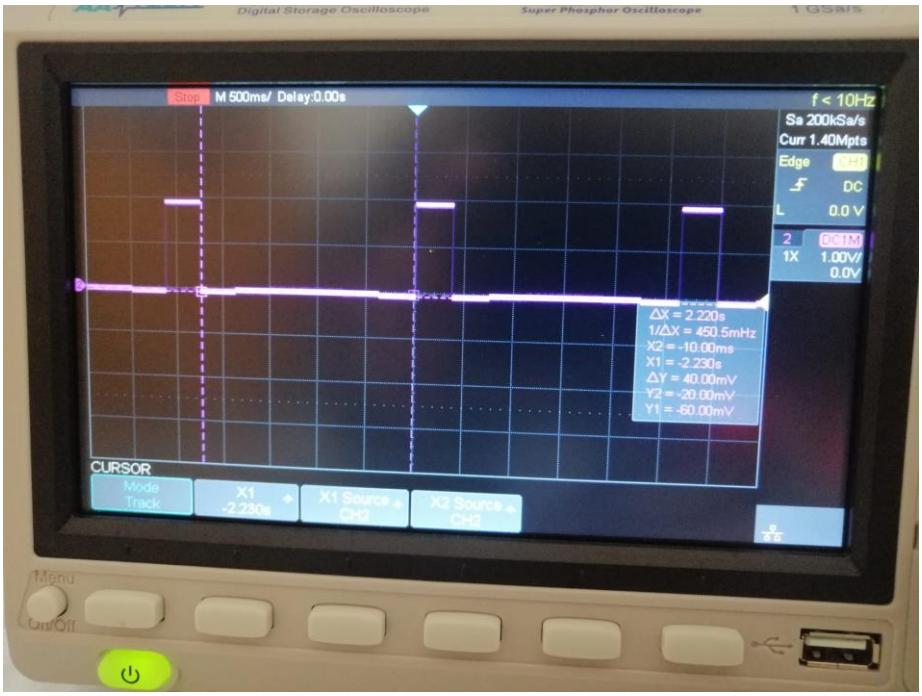


Figure 6: LED1, off time

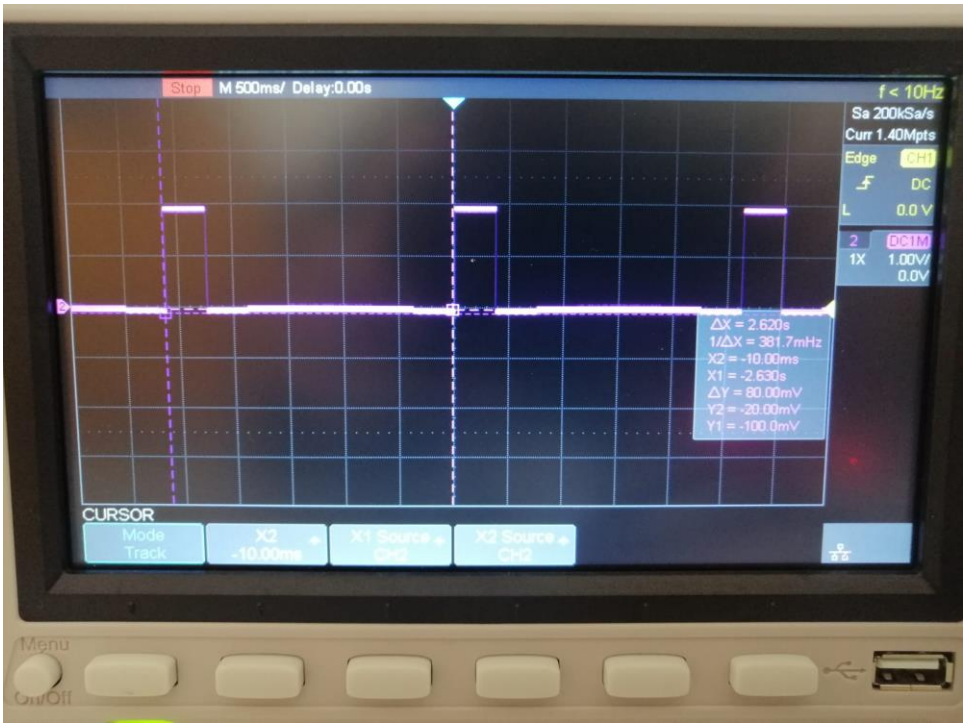


Figure 7: LED1, one cylce period time

- 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.

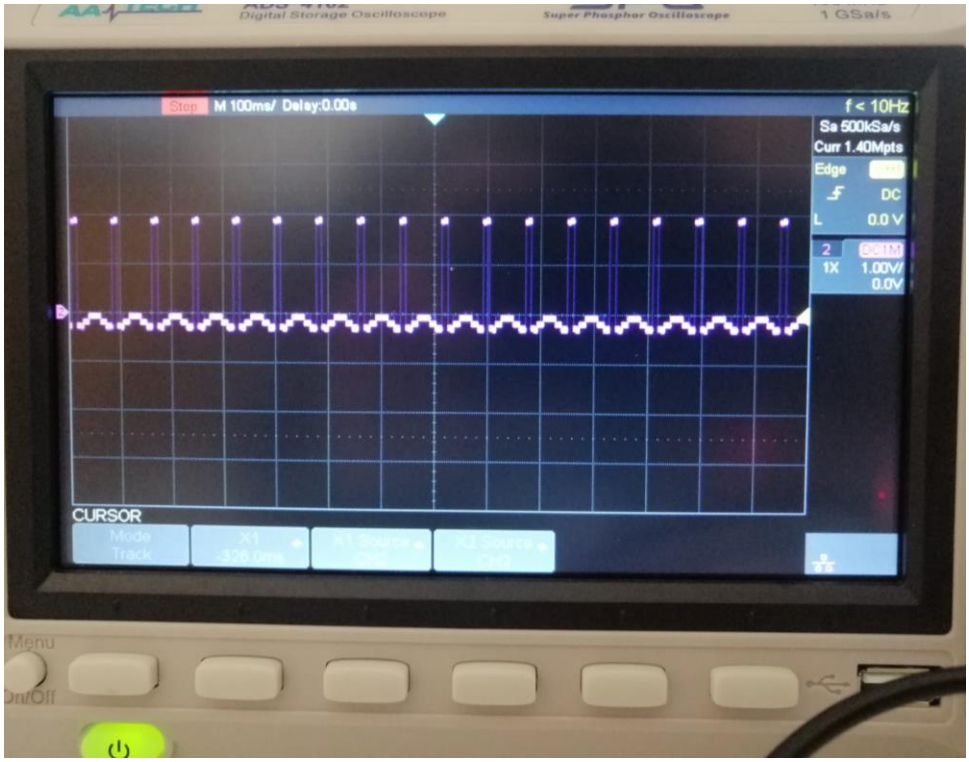


Figure 8: LED1 measurement, delay time=10 ms

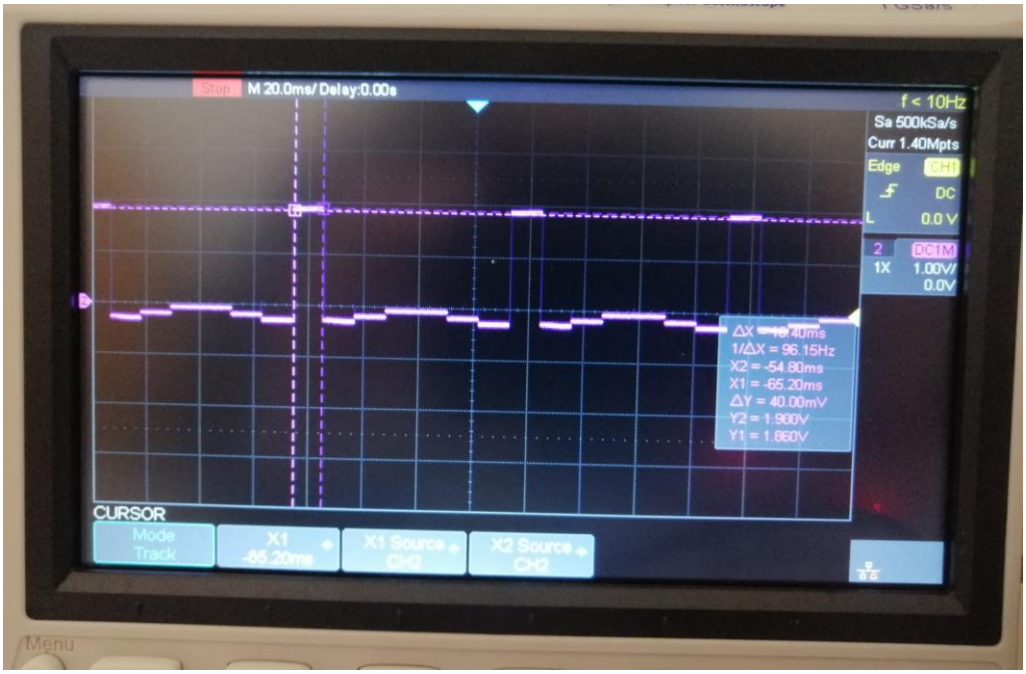


Figure 9: LED1, mesturede on time as 10ms

- There is a latency of 4 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.

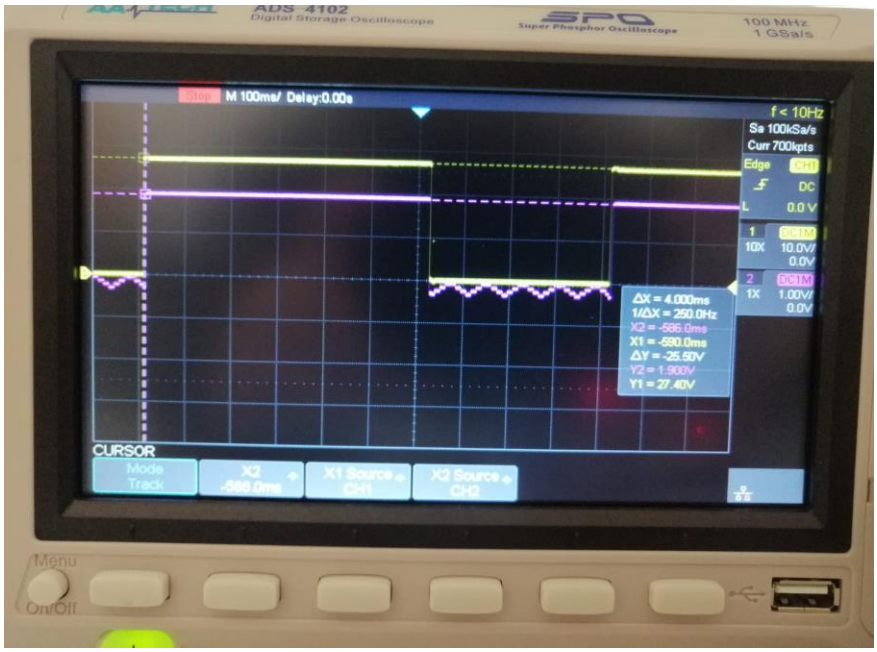


Figure 10: Button and status LED signals, pause latency

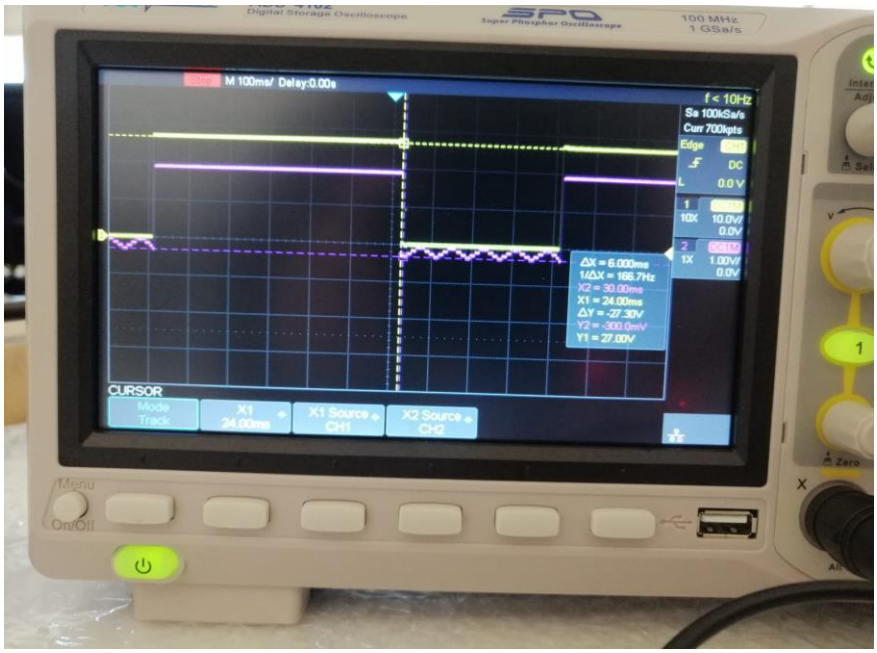


Figure 11: Button and status LED signals, play latency

QUESTION 2

2.i) Assembly Code

```
/*
 * lab2_prob2.s
 *
 *
 */

.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 =DR
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

//Delay Interval
.equ delayInterval, 1000000
```

```
// 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, 0x3
     orrs r5, r5, r4
     str r5, [r6]

     // setup PA8, PA9, PA10 and PA15 for 01 in MODER
     ldr r6, =GPIOA_MODER
     ldr r5, [r6]
     // cannot do with movs, so use pc relative
     ldr r4, =[0x7FD50000] //All PA pins used define output
     ands r5, r5, r4
     str r5, [r6]

     // setup PB0, PB1, PB2 ....PB9 for 01 and PB5 for 00 in
MODER
     ldr r6, =GPIOB_MODER
     ldr r5, [r6]
     // cannot do with movs, so use pc relative
     ldr r4, =[0x55055] //PB5 pin define input, others used
pins define output
     ands r5, r5, r4
     str r5, [r6]

     //D1 Active
     ldr r6, =GPIOA_ODR
     ldr r5, [r6]
     ldr r4, =[0x0700]
     orrs r5, r5, r4
     str r5, [r6]
```

```
     movs r3, [0x0] //Register used for define which pins set
high
     movs r2, [0x0] //Register used for understand which
state is program

Loop:
     ldr r6, = GPIOB_IDR
     ldr r5, [r6] //For PB5, Pass Button
     ldr r7, [r6] //For PB4, Countdown button
     movs r4, #0x20 //Status switch connected to PB5
     ands r5, r5, r4 //Getting the value of button pressed or
not
     lsrs r5, #5 //Shifting to lsb for compare
     cmp r5, #0x1 //Compare IDR Value with 1 bit
     beq changeNumber //If equal
     movs r4, #0x10 //Status switch connected to PB4
     ands r7, r7, r4 //Getting the value of button pressed or
not
     lsrs r7, #4 //Shifting to lsb for compare
     cmp r7, #0x1 //Compare IDR Value with 1 bit
     beq countdown //If equal

     ldr r1, =delayInterval

Delay: //Delay for program work slowly
     subs r1, r1, #1
     bne Delay

     b Loop

countdown:
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x8] //For set PB3 high, status led
     orrs r5, r5, r4
     str r5, [r6]
     cmp r2, [0x1]
     beq FirstCountdown //Countdown for first number
     cmp r2, [0x2]
     beq SecondCountdown //Countdown for second number
     cmp r2, [0x0]
     beq ThirdCountdown //Countdown for third number
     bne CCountinue

FirstCountdown:
     movs r3, [0x0] //Because of first number is 1, just
display 0
     bl NumberSelect //Display number sent
     ldr r1, =delayInterval //Add delay for see transition
```

```
Delay1:
     subs r1, r1, #1
     bne Delay1
     b CCountinue

SecondCountdown:
     movs r3, [0x6]
     bl NumberSelect
     ldr r1, =delayInterval

DelayM:
     subs r1, r1, #1
     bne DelayM
     movs r3, [0x5]
     bl NumberSelect
     ldr r1, =delayInterval

DelayS:
     subs r1, r1, #1
     bne DelayS
     movs r3, [0x4]
     bl NumberSelect
     ldr r1, =delayInterval

Delay2:
     subs r1, r1, #1
     bne Delay2
     movs r3, [0x3]
     bl NumberSelect
     ldr r1, =delayInterval

Delay3:
     subs r1, r1, #1
     bne Delay3
     movs r3, [0x2]
     bl NumberSelect
     ldr r1, =delayInterval

Delay4:
     subs r1, r1, #1
     bne Delay4
     movs r3, [0x1]
     bl NumberSelect
     ldr r1, =delayInterval

Delay5:
     subs r1, r1, #1
     bne Delay5
     movs r3, [0x0]
     bl NumberSelect
     ldr r1, =delayInterval
```

```
Delay6:
     subs r1, r1, #1
     bne Delay6
     b CCountinue

changeNumber:
     cmp r2, [0x0]
     beq FirstNumber
     cmp r2, [0x1]
     beq SecondNumber
     cmp r2, [0x2]
     beq ThirdNumber
     bne CNCountinue


ThirdCountdown:
     movs r3, [0x8]
     bl NumberSelect
     ldr r1, =delayInterval

Delay7:
     subs r1, r1, #1
     bne Delay7
     movs r3, [0x7]
     bl NumberSelect
     ldr r1, =delayInterval

Delay8:
     subs r1, r1, #1
     bne Delay8
     movs r3, [0x6]
     bl NumberSelect
     ldr r1, =delayInterval

Delay9:
     subs r1, r1, #1
     bne Delay9
     movs r3, [0x5]
     bl NumberSelect
     ldr r1, =delayInterval

Delay10:
     subs r1, r1, #1
     bne Delay10
     movs r3, [0x4]
     bl NumberSelect
     ldr r1, =delayInterval

Delay11:
     subs r1, r1, #1
```

```
     bne Delay11
     movs r3, [0x3]
     bl NumberSelect
     ldr r1, =delayInterval

Delay12:
     subs r1, r1, #1
     bne Delay12
     movs r3, [0x2]
     bl NumberSelect
     ldr r1, =delayInterval

Delay13:
     subs r1, r1, #1
     bne Delay13
     movs r3, [0x1]
     bl NumberSelect
     ldr r1, =delayInterval

Delay14:
     subs r1, r1, #1
     bne Delay14
     movs r3, [0x0]
     bl NumberSelect
     ldr r1, =delayInterval

Delay15:
     subs r1, r1, #1
     bne Delay15
     b CCountinue

CCountinue:
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x8]
     bics r5, r5, r4
     str r5, [r6]
     b Loop

FirstNumber:
     movs r3, [0x1]
     bl NumberSelect
     movs r2, [0x1]
     b CNCountinue

SecondNumber:
     movs r3, [0x7]
     bl NumberSelect
     movs r2, [0x2]
     b CNCountinue
```

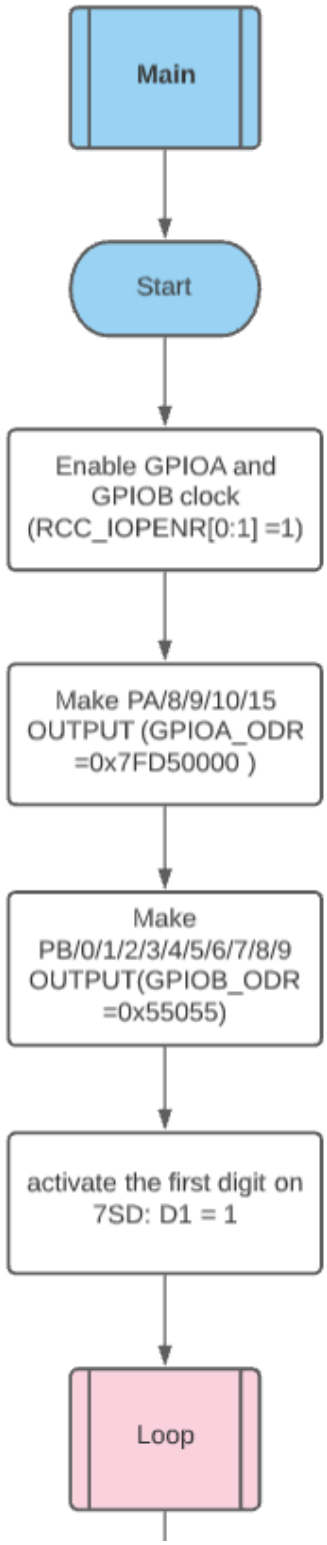
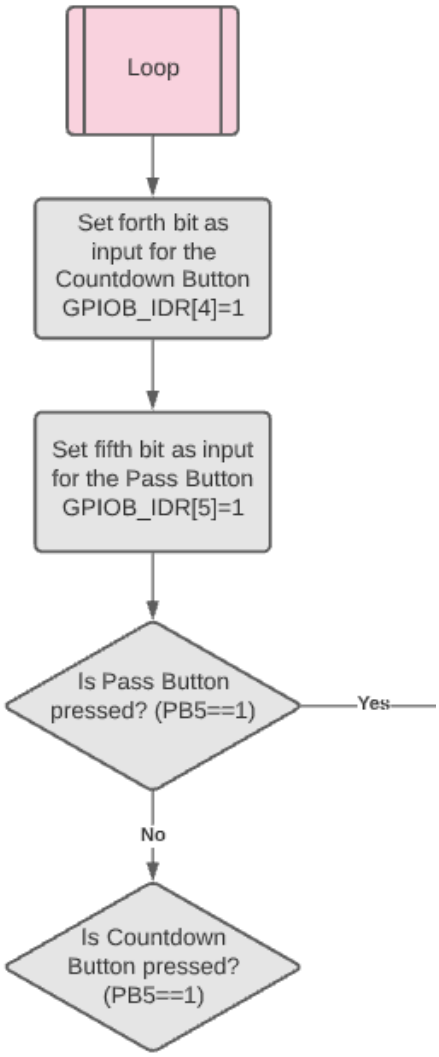
```
ThirdNumber:
     movs r3, [0x9]
     bl NumberSelect
     movs r2, [0x0]
     b CNCountinue

CNCountinue:
     b Loop


NumberSelect:
     cmp r3, [0x0] //Control r3 for which number sent to
NumberSelect
     beq NumberZero
     cmp r3, [0x1]
     beq NumberOne
     cmp r3, [0x2]
     beq NumberTwo
     cmp r3, [0x3]
     beq NumberThree
     cmp r3, [0x4]
     beq NumberFour
     cmp r3, [0x5]
     beq NumberFive
     cmp r3, [0x6]
     beq NumberSix
     cmp r3, [0x7]
     beq NumberSeven
     cmp r3, [0x8]
     beq NumberEight
     cmp r3, [0x9]
     beq NumberNine
     bne NSCountinue

NumberZero: //Display the number sent
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0xFFD]
     bics r5, r5, r4
     str r5, [r6]
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x1C7]
     orrs r5, r5, r4
     str r5, [r6]
     b NSCountinue

NumberOne:
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0xFF7]
```

```
     bics r5, r5, r4
     str r5, [r6]
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x42]
     orrs r5, r5, r4
     str r5, [r6]
     b NSCountinue

NumberTwo:
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0xFF7]
     bics r5, r5, r4
     str r5, [r6]
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x2C5]
     orrs r5, r5, r4
     str r5, [r6]
     b NSCountinue

NumberThree:
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0xFF7]
     bics r5, r5, r4
     str r5, [r6]
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x2C3]
     orrs r5, r5, r4
     str r5, [r6]
     b NSCountinue

NumberFour:
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0xFF7]
     bics r5, r5, r4
     str r5, [r6]
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x342]
     orrs r5, r5, r4
     str r5, [r6]
     b NSCountinue

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

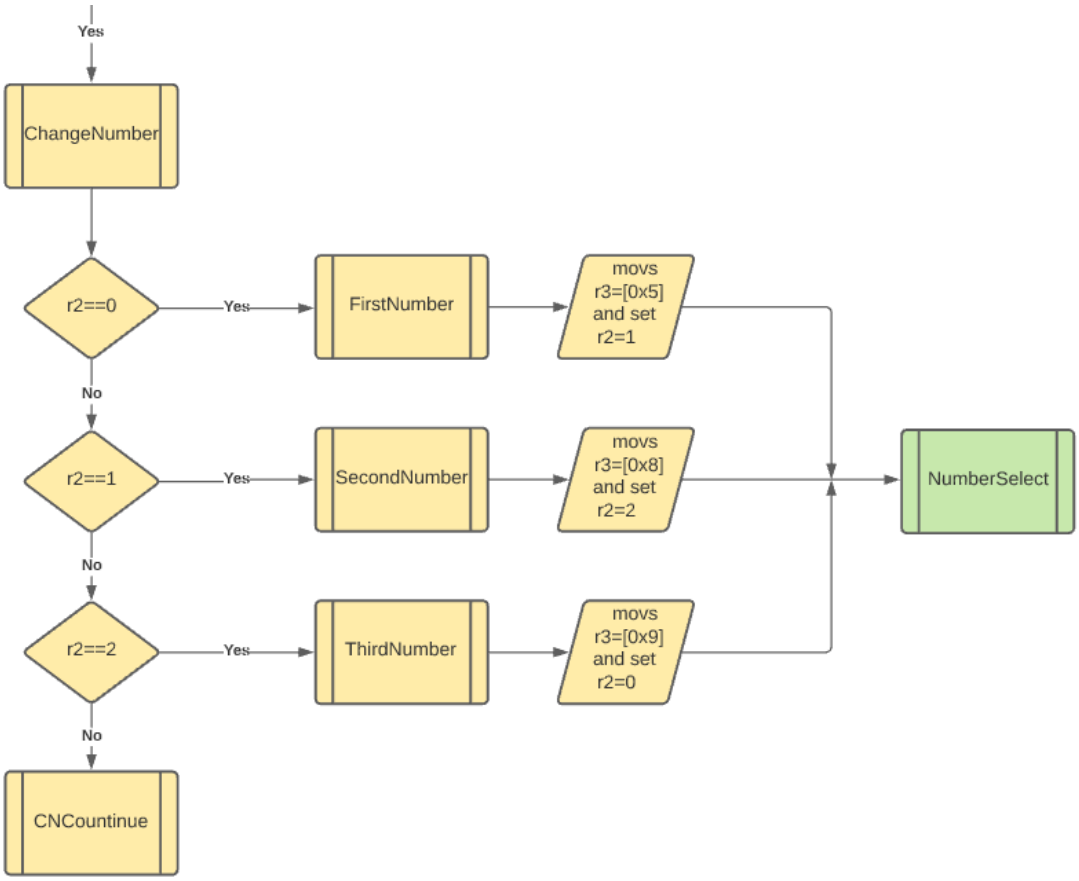
```
     ldr r4, =[0xFF7]
     bics r5, r5, r4
     str r5, [r6]
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x383]
     orrs r5, r5, r4
     str r5, [r6]
     b NSCountinue

NumberSix:
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0xFF7]
     bics r5, r5, r4
     str r5, [r6]
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x387]
     orrs r5, r5, r4
     str r5, [r6]
     b NSCountinue

NumberSeven:
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0xFF7]
     bics r5, r5, r4
     str r5, [r6]
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0xC2]
     orrs r5, r5, r4
     str r5, [r6]
     b NSCountinue

NumberEight:
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0xFF7]
     bics r5, r5, r4
     str r5, [r6]
     ldr r6, =GPIOB_ODR
     ldr r5, [r6]
     ldr r4, =[0x3C7]
     orrs r5, r5, r4
     str r5, [r6]
     b NSCountinue

NumberNine:
     ldr r6, =GPIOB_ODR
```

```
      ldr r5, [r6]
      ldr r4, =[0xFF7]
      bics r5, r5, r4
      str r5, [r6]
      ldr r6, =GPIOB_ODR
      ldr r5, [r6]
      ldr r4, =[0x3C3]
      orrs r5, r5, r4
      str r5, [r6]
      b NSCountinue

NSCountinue:
      bx lr


      // this should never get executed
      nop
```

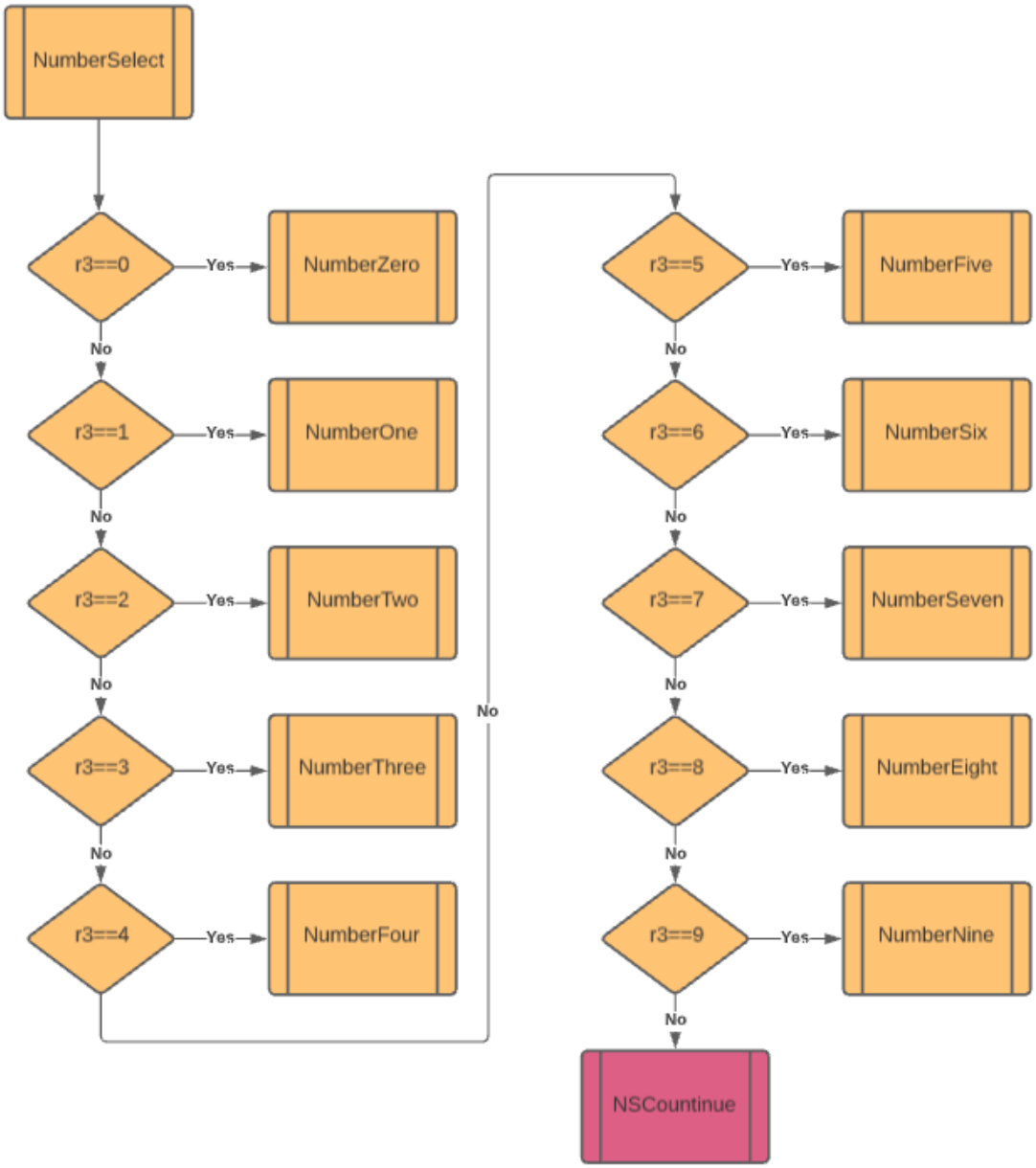
2.ii) Flowchart

1- Initiation: clock enable, Set MODER, accesing Loop	**2-** Main Loop, If button pressed go Change Number function
	

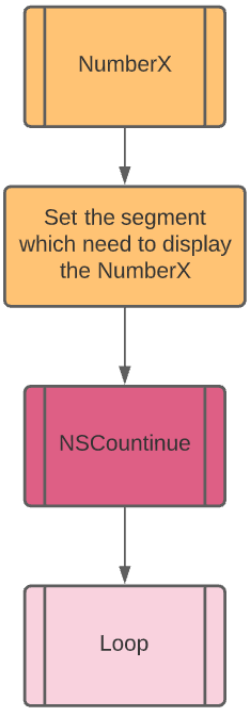
3- Change Number Function: Change between student number's last digit



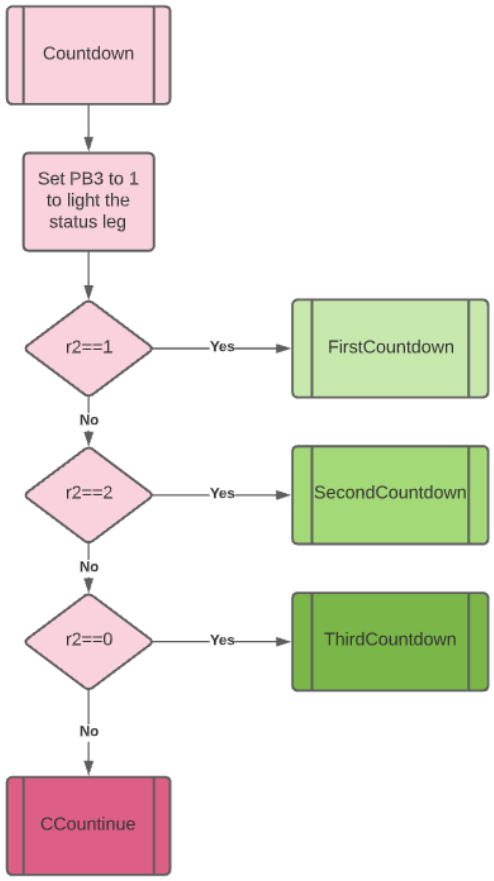
4- Number Select Function: goes NumberX function depends on the register



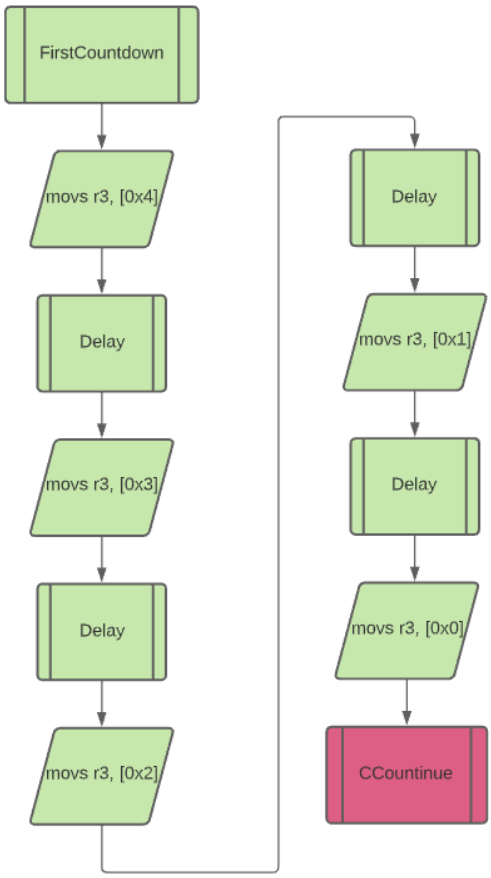
5- NumberX functions: that functions light up segments as a number (X = 1, 2, 3, 4, 5, 6, 7, 8, 9)



6- Countdown Function: Changes countdown number as push button

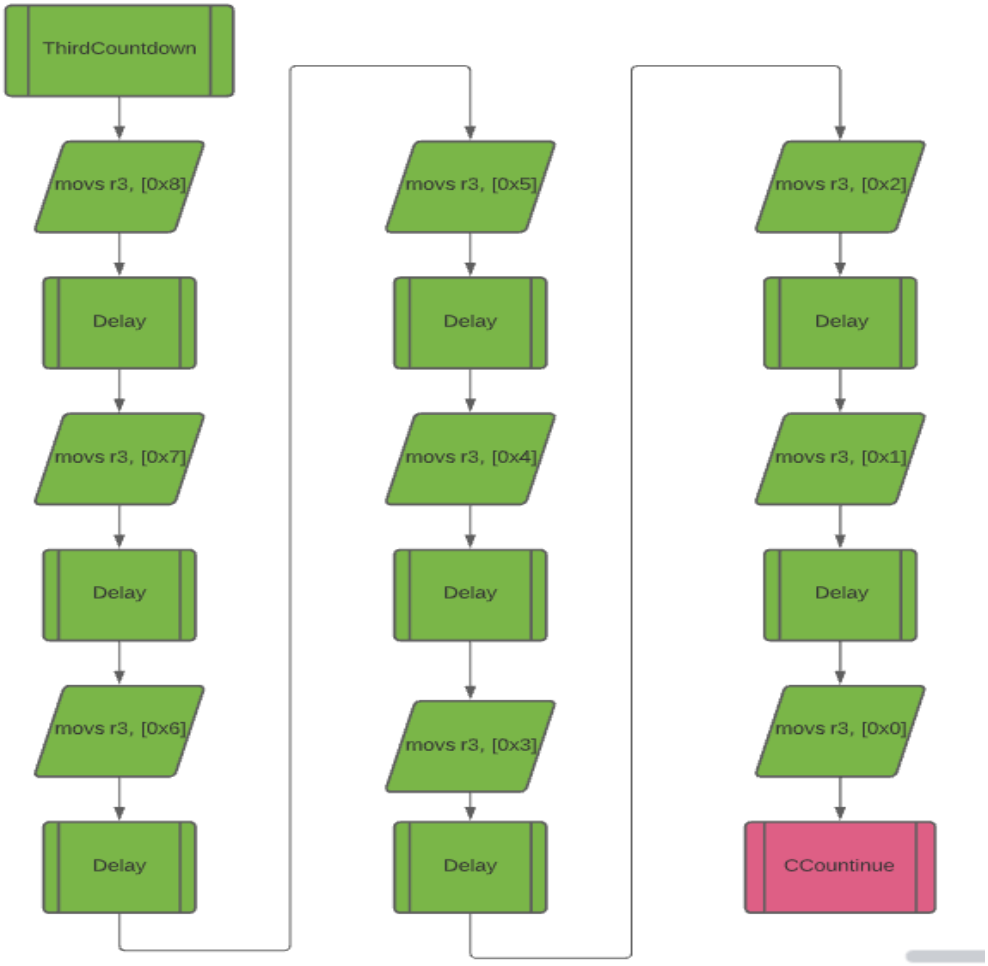


7- FirstCountDown Function: goes down first student's last digit

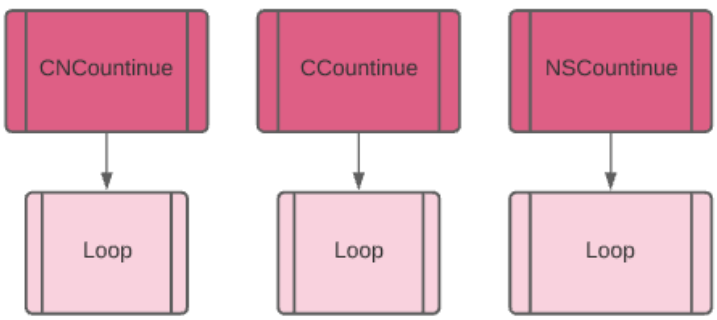


8- SecondCountDown Function: goes down second student's last digit

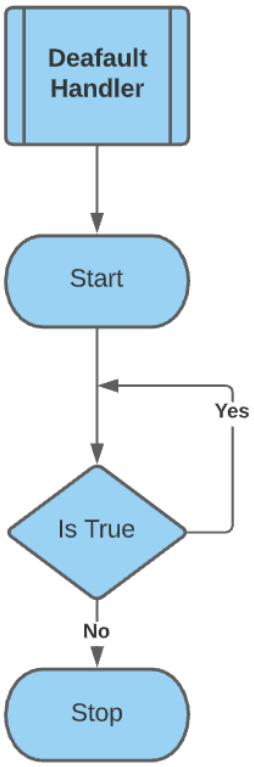
9- ThirdCountDown Function: goes down third student's last digit



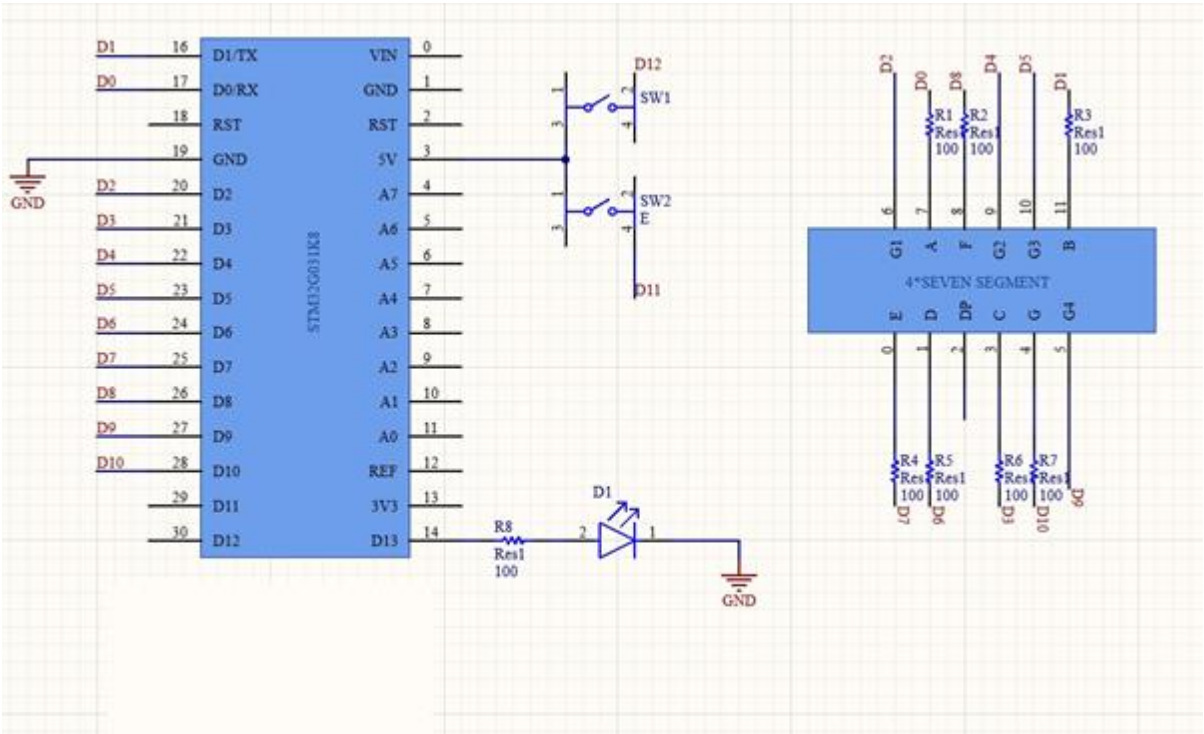
10- All functions go for a loop after operation.



11- If something happens default handler waits to catch branch.



2.iii) Circuit Scheme



2.iv) Oscilloscope Measurements

- The on time satisfied the required time of one second.

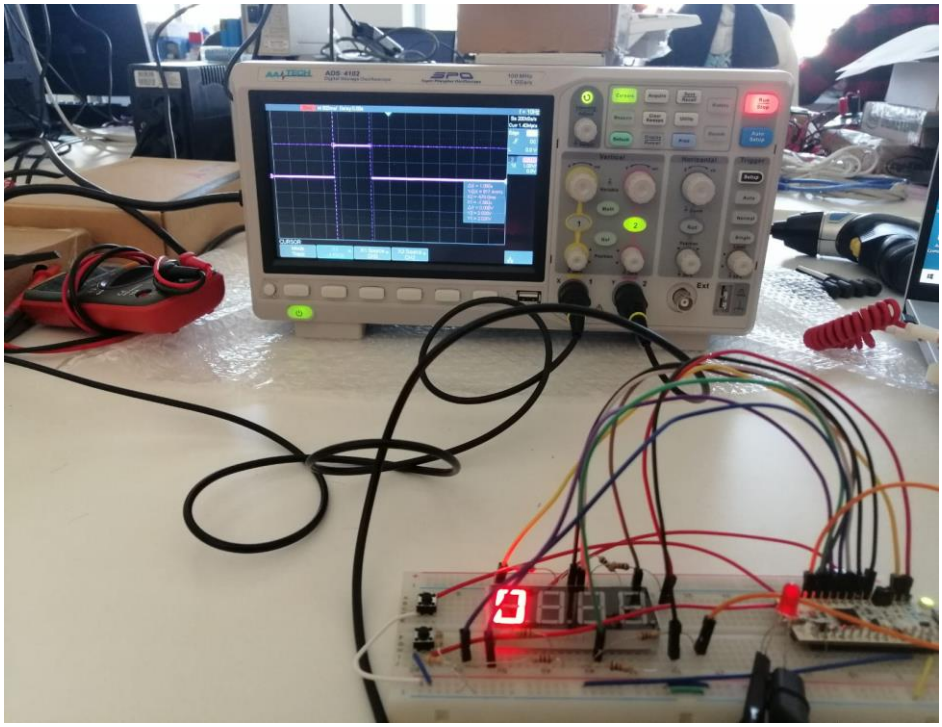


Figure 12: 7SD, after a cycle

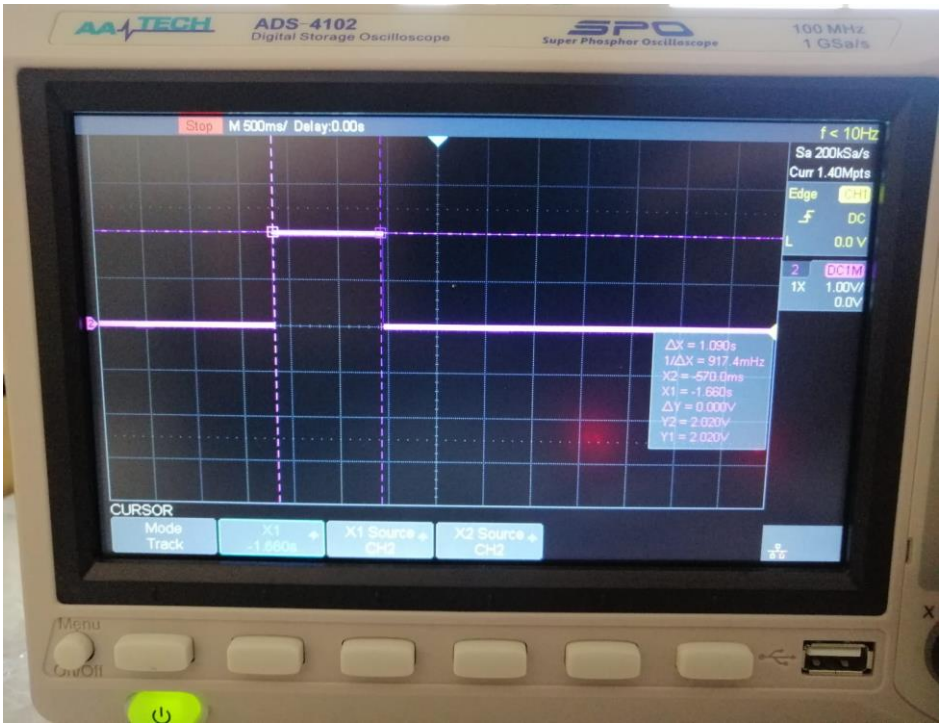


Figure 13: Status LED, on time

- We have set the button with a pull down resistor since, it pulls the signal to zero we could not observe bouncing. Not all the buttons needs debouncing. The only bouncing that needs to be fixed is the button that changes the last digit of the school number because when we pressed the button it could bounce and skip to person in a one press. The other button that starts the program does not need any debouncing because the only thing it needs to do is to start program.