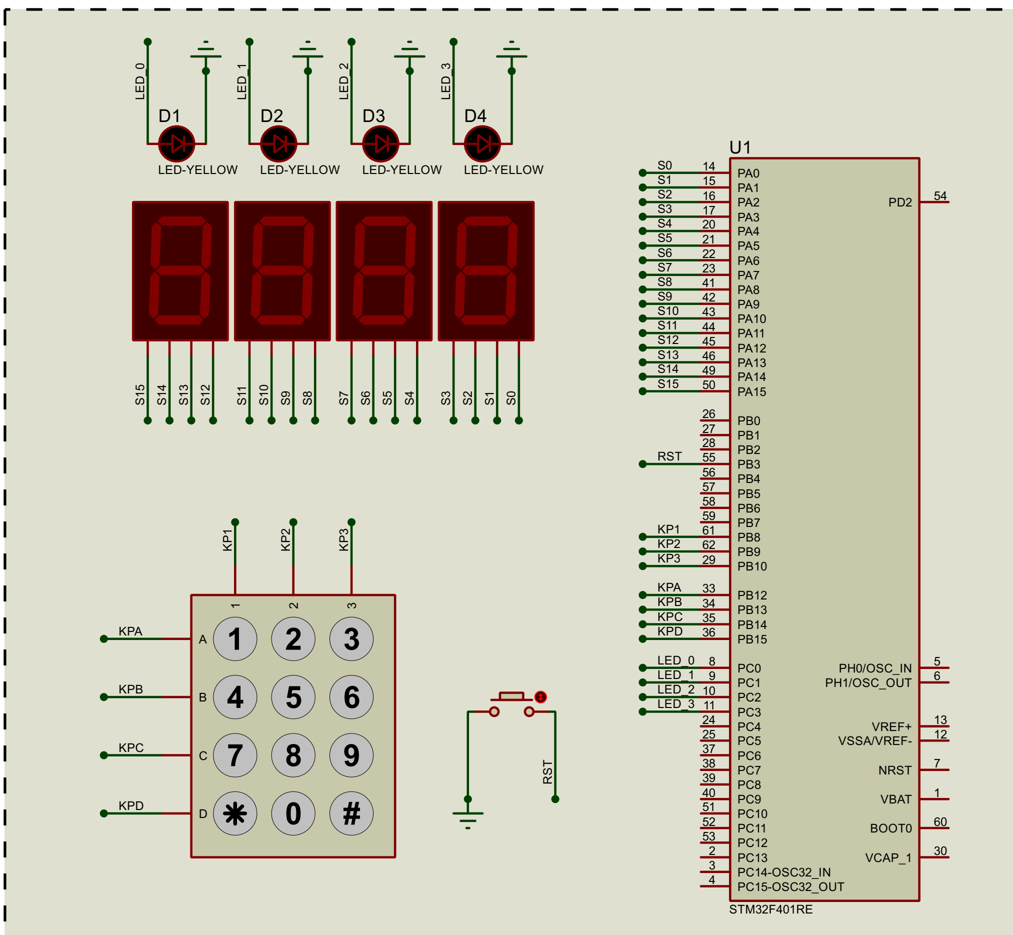
Mohammad Hashemi | 97243073
Instructor’s Name: Dr. Dara Rahmati
May 10, 2021

HW06 Report

Microprocessor and Assembly Language Course - Spring 2021

Elevator Display Simulator using STM32F401RE



1. Memory Map

Since we used STM32F401RE, we must config its ports. As we know there are three ports in it: A, B and C. We will use them as GPIOs. Therefore we should refer to its data sheet to find the addresses of the desired ports:

Table 1. STM32F401xB/C and STM32F401xD/E register boundary addresses

Boundary address	Peripheral	Bus	Register map
0x5000 0000 - 0x5003 FFFF	USB OTG FS	AHB2	*Section 22.16.6: OTG_FS register map on page 755*
0x4002 6400 - 0x4002 67FF	DMA2	AHB1	*Section 9.5.11: DMA register map on page 198*
0x4002 6000 - 0x4002 63FF	DMA1		
0x4002 3C00 - 0x4002 3FFF	Flash interface register		*Section 3.8: Flash interface registers on page 60*
0x4002 3800 - 0x4002 3BFF	RCC		*Section 6.3.22: RCC register map on page 137*
0x4002 3000 - 0x4002 33FF	CRC		*Section 4.4.4: CRC register map on page 70*
0x4002 1C00 - 0x4002 1FFF	GPIOH		*Section 8.4.11: GPIO register map on page 164*
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

The address space which we need is shown within a red box in the table above. Each of GPIOA, GPIOB and GPIOC has its own registers which should be configure based on their usages. We want to use them as "input", "output". So the following register values should be modified:

1. GPIO port mode register (GPIOx_MODER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

The GPIOA pins will be used as output(7-Segments), GPIOB pins will be used as both output(Keypad rows) and input(Reset push-button) and GPIOC pins will be used as output(LEDs).

2. GPIO port input data register (GPIOx_IDR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)
These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

will be discussed later.

3. GPIO port output data register (GPIOx_ODR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)
These bits can be read and written by software.
Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..E and H).

will be discussed later.

These addresses are stored in the following variables to use later in the code:

```
;Addresses
;------------------------------------
; Clock
RCC_AHB1ENR       EQU 0x40023830

; GPIOA - Base Addr: 0x4002 0000
GPIOA_MODER       EQU 0x40020000
GPIOA_IDR         EQU 0x40030010
GPIOA_ODR         EQU 0x40020014

; GPIOB - Base Addr: 0x4002 0400
GPIOB_MODER       EQU 0x40020400
GPIOB_IDR         EQU 0x40020410
GPIOB_ODR         EQU 0x40020414

; GPIOC - Base Addr: 0x4002 0800
GPIOC_MODER       EQU 0x40020800
GPIOC_IDR         EQU 0x40020810
GPIOC_ODR         EQU 0x40020814

;------------------------------------
```

2. System Initialization

The usages of each register was discussed in the previous part. We must initialize the registers in the ARM source code. Firstly, the clock of each register should be turned on. The following table is showing the RCC AHB1 peripheral reset register (RCC_AHB1RSTR) which will be used for the discussed purpose:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									DMA2 RST	DMA1 RST	Reserved				
									rw	rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCRST	Reserved				GPIOH RST	Reserved		GPIOE RST	GPIOD RST	GPIOC RST	GPIOB RST	GPIOA RST
			rw					rw			rw	rw	rw	rw	rw

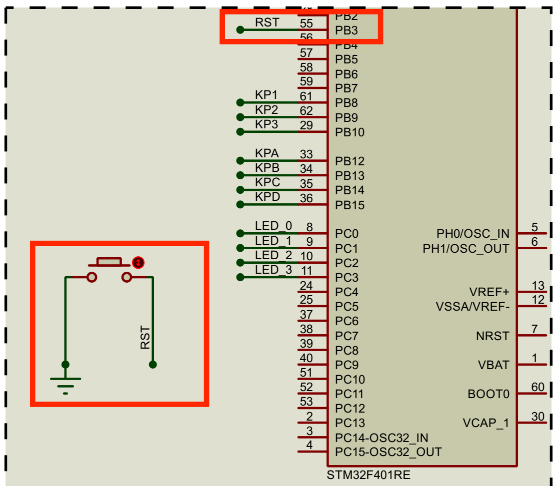
The red box shows us that the least three bits in the RCC register should be set to 1. The others don't care!

```
SystemInit FUNCTION

      ; Enable clk FOR portA, portB AND portC
      LDR   R1, =RCC_AHB1ENR
      MOV   R0, #0x7 ; =000...0111
      STR   R0, [R1]

      ; GPIOA MODER -> All the pins are used as output
      LDR   R1, =GPIOA_MODER
      LDR   R0, [R1]
      MOV   R0, #0x55555555 ; =010101...0101
      STR   R0, [R1]

      ; GPIOB MODER -> Some for output and some for input
      LDR   R1, =GPIOB_MODER
      LDR   R0, [R1]
      MOV   R0, #0x55005500
      STR   R0, [R1]

      ; GPIOC MODER
      LDR   R1, =GPIOC_MODER -> the least four pins are used for output
      LDR   R0, [R1]
      MOV   R0, #0x00000055
      STR   R0, [R1]

      ENDFUNC
```

3. Main function

The program is implemented by using two main loops. The first one is for getting the source and destination levels of the building from the user via keypad. In the following parts, each component implementation will be discussed separately:

3.1. Reset push-button handler

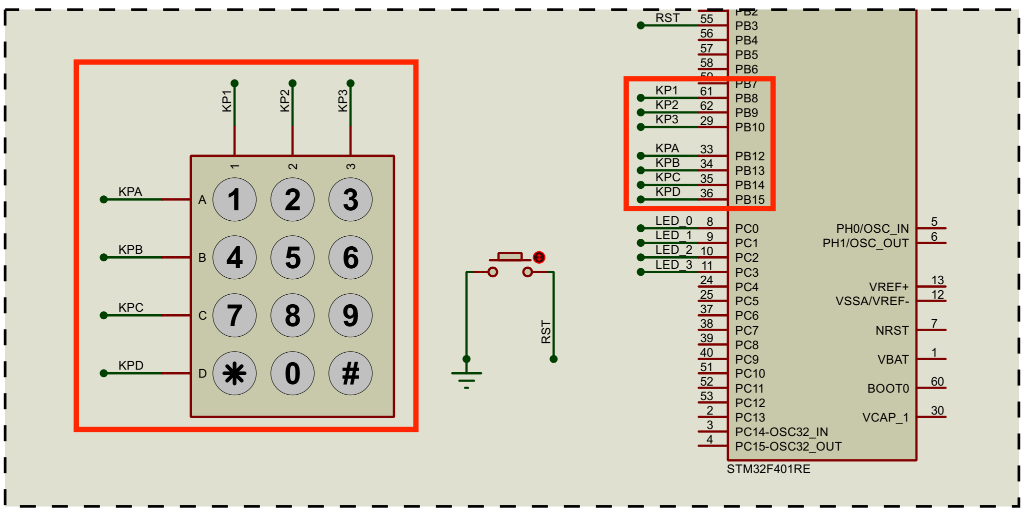


The PB3 is the pin(input) used for push-button IO operation. So we read the value of GPIOB_IDR register and compare it with #0xFFF7:

```
MAIN_LOOP


    ;-------------------------------------
    ; Check if the reset button is pressed
    LDR R1, =GPIOB_IDR
    LDR R0, [R1]
    MOV R1, R0

    MOV R2, #0xFFF7 ; 1111 1111 1111 0111
    ORR R0, R0, R2
    CMP R0, R2
    BEQ RESET
    ;-------------------------------------
```

```
RESET
        LDR R1, =GPIOA_ODR
        MOV R0, #0
        STR R0, [R1]
        MOV R6, #0
        MOV R7, #0
        B MAIN_LOOP
```

3.2. Keypad handler

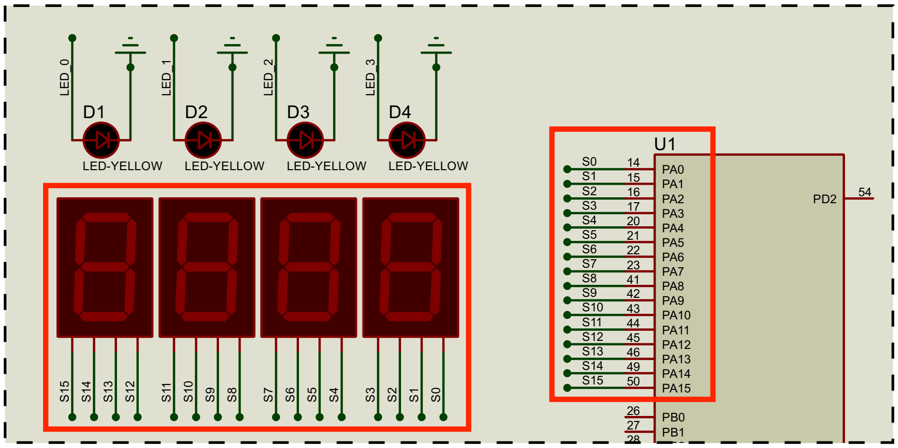


As we have learnt in the TA class, for using the phone-keypad, 4 pins should be used as output(PB12 - PB15) for writing in the rows(A, B, C or D) and three pins as input(PB8 - PB10). The code below shows how we should read and write into GPIOB for the numbers 1, 2 and 3 in the keypad. For the rest of the numbers we can repeat this approach.

```
;-------------------------------------
      ;First row in keypad enable
      LDR   R2, =GPIOB_ODR
      MOV R3, #0xEFFF ; =1110 1111 1111 1111
      STR R3, [R2]

      LDR R2, =GPIOB_IDR
      LDR R0, [R2]
      MOV R1, R0
      ; Check if 1 is pressed
      MOV R4, #0xFEFF   ; =1111 1110 1111 1111
      ORR R0, R0, R4
      CMP R0, R4
      MOV R0, #1
      BEQ DISPLAY_7SEG
      ;Check if 2 is pressed
      MOV R4, #0xFDFF   ; =1111 1101 1111 1111
      MOV R3, R1
      ORR R3, R3, R4
      CMP R3, R4
      MOV R0, #2
      BEQ DISPLAY_7SEG
      ;Check if 3 is pressed
      MOV R4, #0xFBFF   ; =1111 1011 1111 1111
      MOV R5, R1
      ORR R5, R5, R4
      CMP R5, R4
      MOV R0, #3
      BEQ DISPLAY_7SEG
      ;-------------------------------------
```

3.3. Display on the 7-Segment

At the beginning of the program execution, the user must enter two numbers between 0-9. Then they should be displayed at the most left side of the 7-Segments.



As we know, GPIOA is used for outputting into 7-Segments. Since we have 4 7-seg component, we must modify the total pins of GPIOA.

Imagine the user entered 4 as the source level. If we output 4 into the PA0-PA15 then the 7-segments will show 0004 which is not the one we expect. So before outputting, we must shift 12 times the bits to the left to achieve 4000. Here is what the code does:

```
DISPLAY_7SEG
      ; R0 should contain the expected value to be displayed
      LSL R0, R0, #12               ; R0 = R0 * 1000 (in decimal)
      LDR R1, =GPIOA_ODR
      STR R0, [R1]
      CMP R7, R0
      BEQ MAIN_LOOP
      CMP R6, #1
      BEQ COUNTER
      MOV R6, #1
      MOV R7, R0          ; R7 = source
      B MAIN_LOOP
```

In the code above there are some CMP instruction which are to know whether the user has inputted the second input as the destination level or not.

3.4. Counter

As the user inputted the source and destination levels, the code execution entered into the second main loop. It must count!

In this step we have the values of the source and destination level(12 left shifted) in R7 and R0 respectively. For example if the user entered 4 and 8 then we have:

R7 = 0x4000 (source level)

R0 = 0x8000 (destination level)

At first, R7 must be displayed. After that, we must display 0x4500, 0x4560 and finally 0x4567.

So how to achieve 0x4500 from the previous step?

R7 = 0100 0000 0000 0000 (= 0x4000 --> LSL 0x0004 #12times)

R9 = 0000 0101 0000 0000 (=0x0500 --> LSL 0x0005 #8 times)

By ORRing R7 and R9 we will have 0100 01010 0000 0000 (=0x4500) :)

For next iteration we store 0x4500 into the R7 and:

R7 = 0100 0101 0000 0000 (= 0x4500)

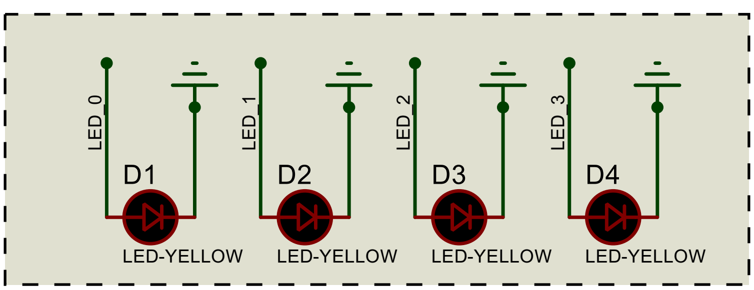
R9 = 0000 0000 0110 0000 (=0x0500 --> LSL 0x0006 #4 times)

And then again by ORRing R7 and R9 we have 0100 01010 0110 0000 (=0x4560).

We repeat this approach again and again until the end. And one note is that if the destination value is less than the source value, instead of incrementing the values, we decrement them. If it is still obscure for you to know how does it work, I highly recommend you to see the source code. (GPIO.s file)

```
START
      CMP R12, #1
      BEQ L2                        ; if destination < source

;-------------------------------------
; For when destination > source
L1
      CMP R5, R6
      BGT END_LOOP
      LSL R9, R5, R8
      ORR R7, R7, R9
      ADD R5, R5, #1
      SUB R8, R8, #4
      B DISPLAY
;-------------------------------------
; For when destination < source
L2
      CMP R5, R6
      BLT END_LOOP
      LSL R9, R5, R8
      ORR R7, R7, R9
      SUB R5, R5, #1
      SUB R8, R8, #4
;-------------------------------------
```

3.5. LEDs

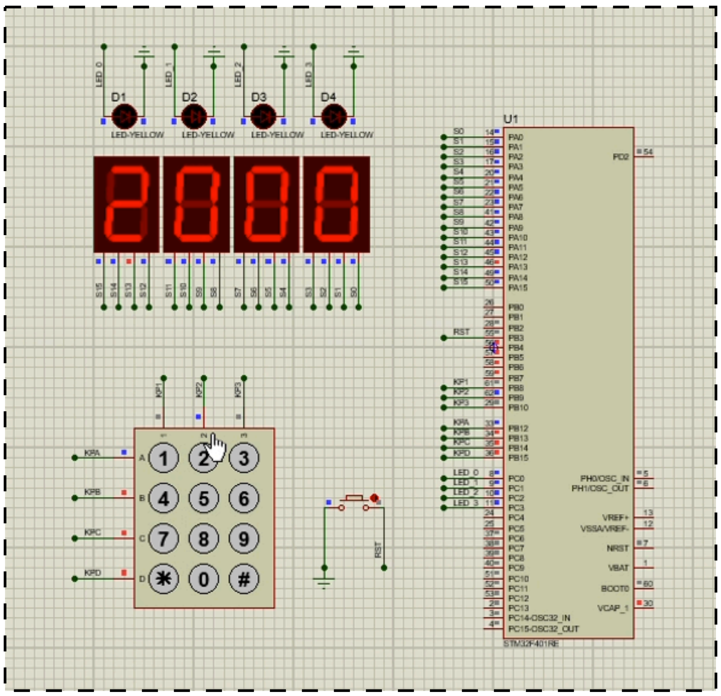


GPIOC is used for outputting into the LEDs(PC0-PC3).

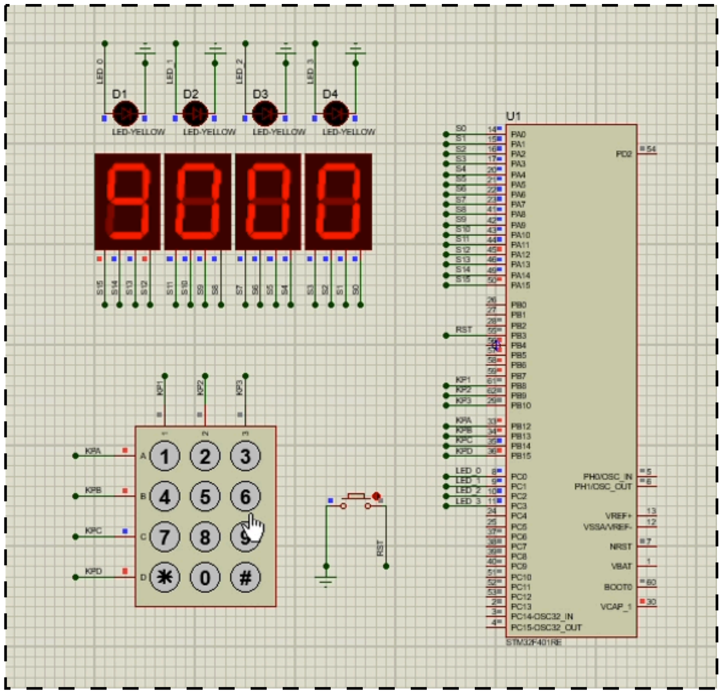
LED	Register Value
LED_0	0x0001 = 0000 0000 0000 0001
LED_1	0x0002 = 0000 0000 0000 0010
LED_2	0x0004 = 0000 0000 0000 0100
LED_3	0x0008 = 0000 0000 0000 1000

```
;-------------------------------------
      ; TURN THE CORRESPONDING LED ON
      LDR R1, =GPIOC_ODR
      STR R11, [R1]
      LSL R11, R11, #1
      ;-------------------------------------
```

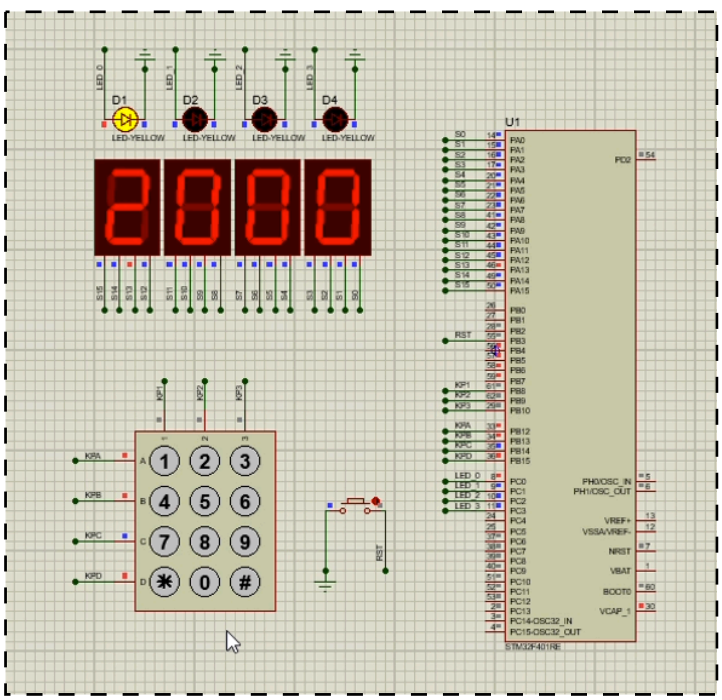
4. Some snap shots of the program execution

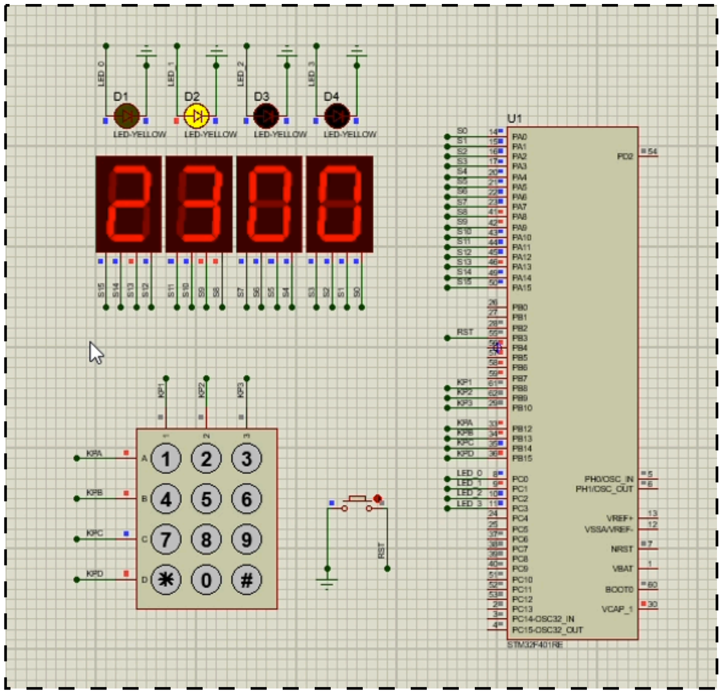


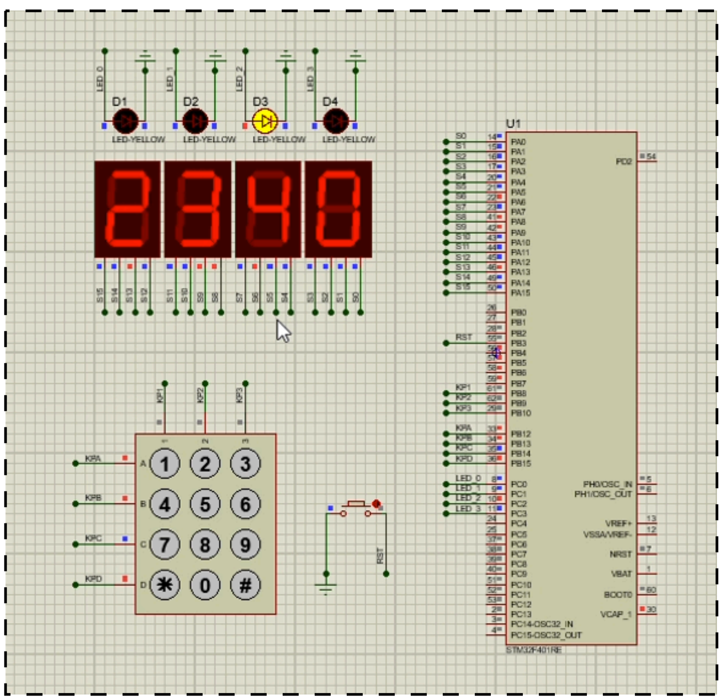
User input(source): 2

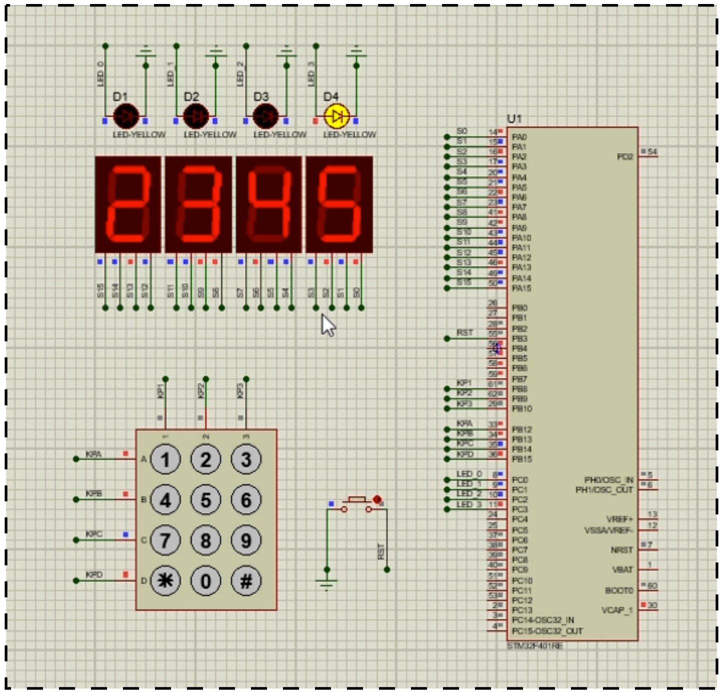


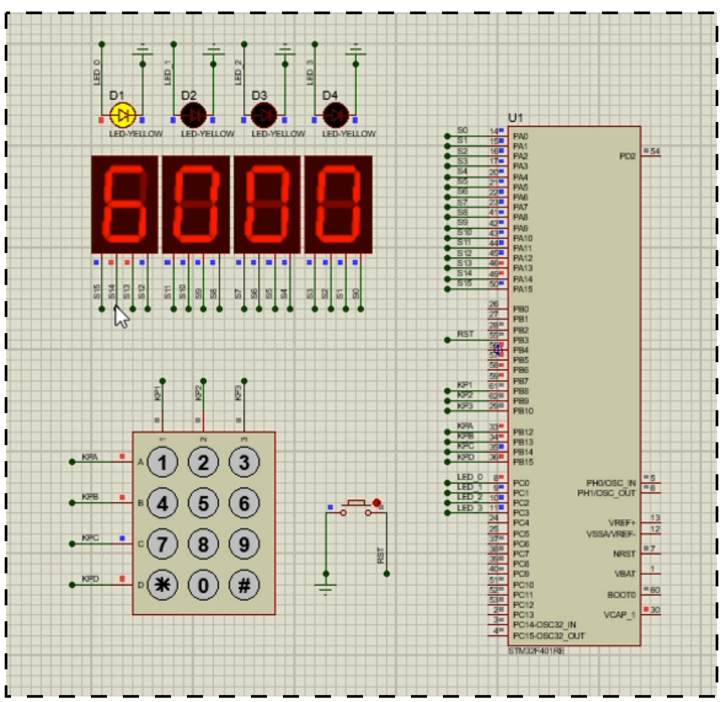
User input(destination): 9

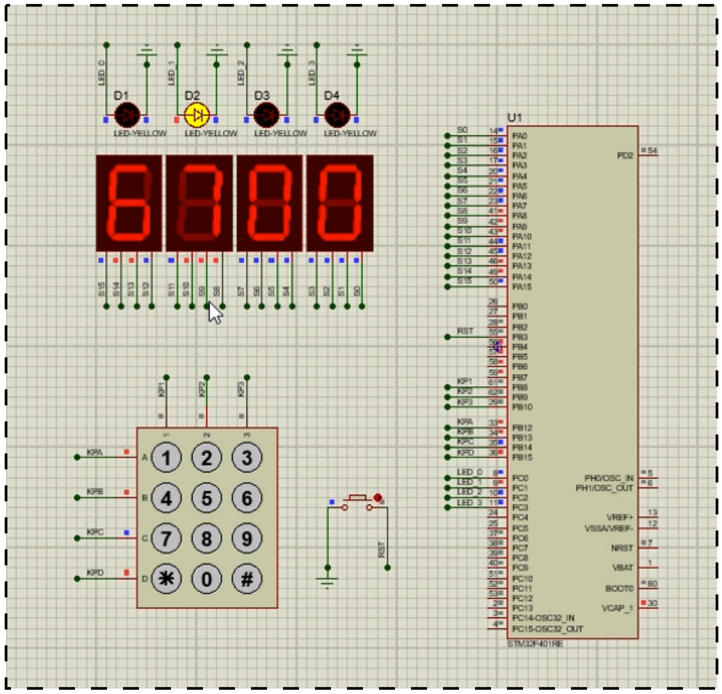


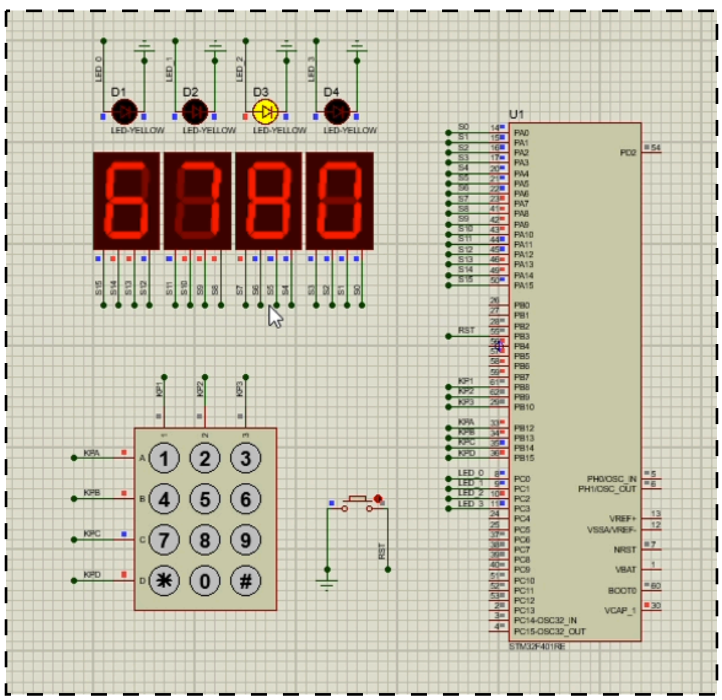

D1
D2
D3
D4
LED-YELLOW
U1
1 2 3
4 5 6
7 8 9
* 0 #

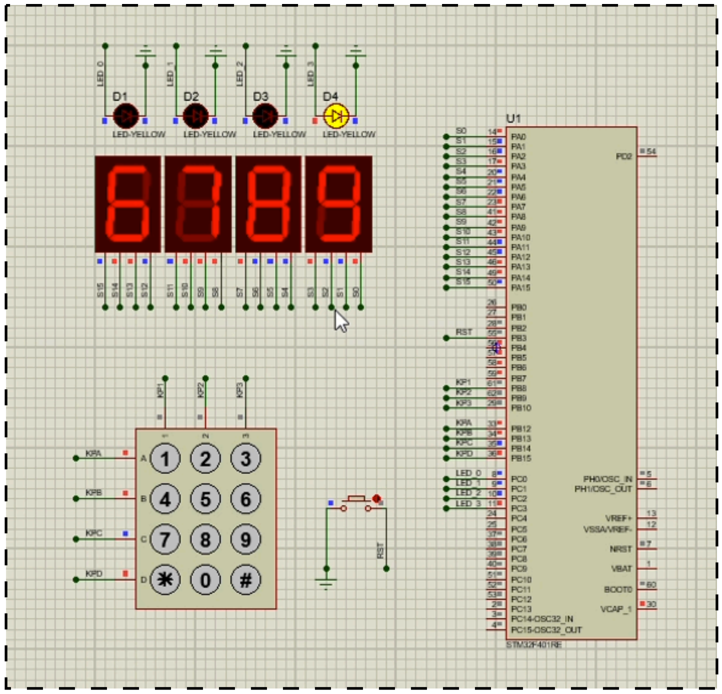

D1
D2
D3
D4
LED-YELLOW
U1
1 2 3
4 5 6
7 8 9
* 0 #











There is a gif in the project directory which shows this execution(demo.gif). Watch it :)