GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRONICS ENGINEERING

ELEC 335

Microprocessors Laboratory

Lab #2 Experiment Report

Hazırlayanlar
200102002025 – Umut Mehmet ERDEM
200102002051 – Arda DERİCİ
200102002061 – Serdar BAŞYEMENİCİ

1. Introduction

In this ELEC 335 laboratory 2 experiment, assembly codes are written that solve the following problems:

- toggles the onboard LED at a rate of 1 second
- controlling LED flashing with a button
- enables toggle of 8 LEDs at the same time at a rate of 1 second
- implements shift pattern with 8 LEDs and the button
- implements a “Knight Rider (Kara Şimşek)” with 8 LEDs

Assembly codes written on STMCubeIDE are transferred to STM32 NUCLEO-G031K8 using a micro USB cable. Flow chart and schematic diagram are drawn for each problem with draw.io.

2. Problems

2.1. Problem I

2.1.1. Flow Chart and Schematic Diagram

The flow chart of Problem I is as follows.

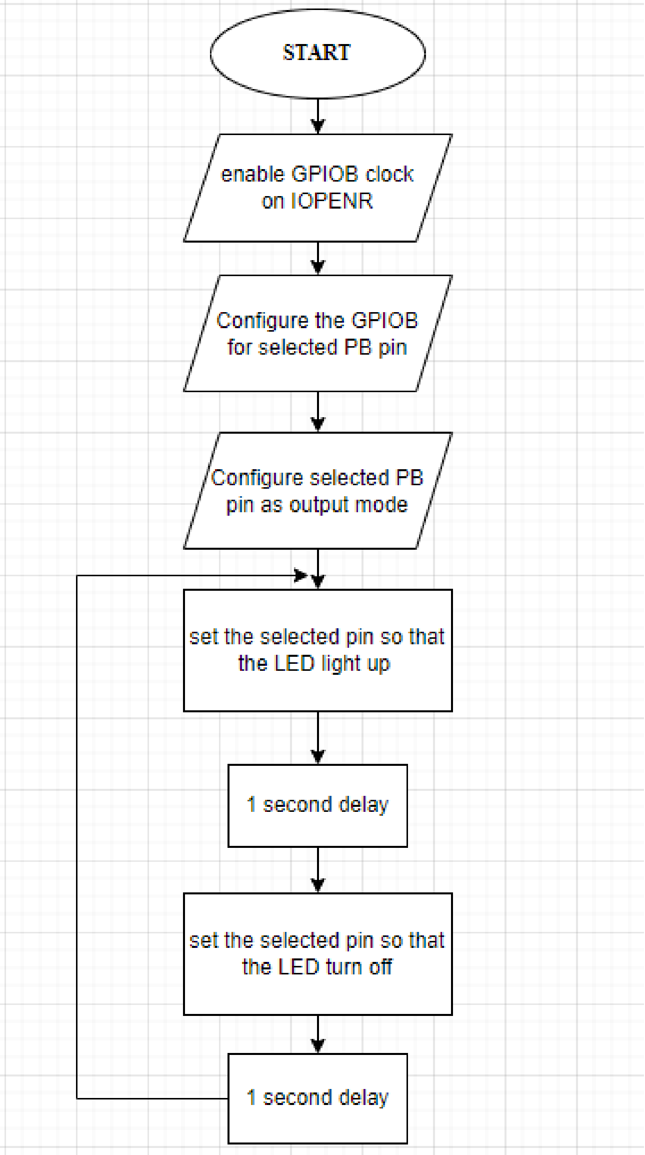
START
enable GPIOB clock on IOPENR
Configure the GPIOB for selected PB pin
Configure selected PB pin as output mode
set the selected pin so that the LED light up
1 second delay
set the selected pin so that the LED turn off
1 second delay

Figure 1 - flow chart for Problem I

The schematic diagram of Problem I is as follows.

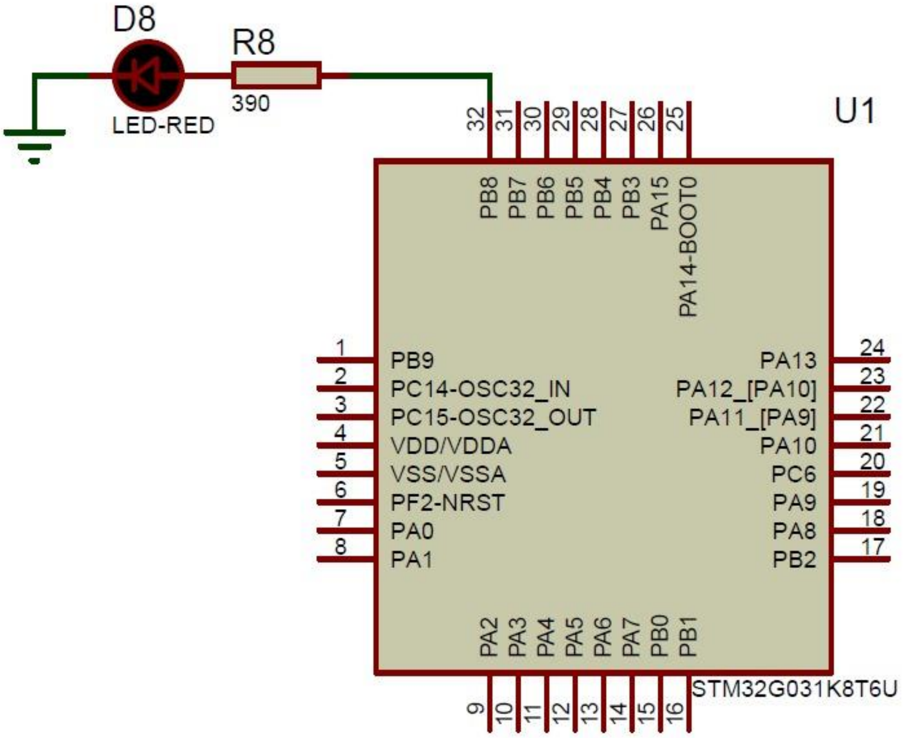

Figure 2 - *schematic diagram for Problem I*

2.1.2. Installation and Operation of the Circuit

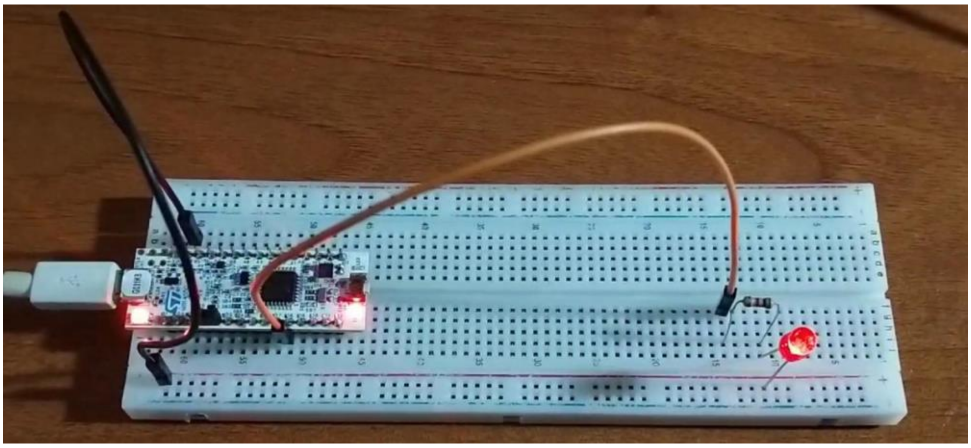

Figure 3 - on the board LED that will toggle is turn on

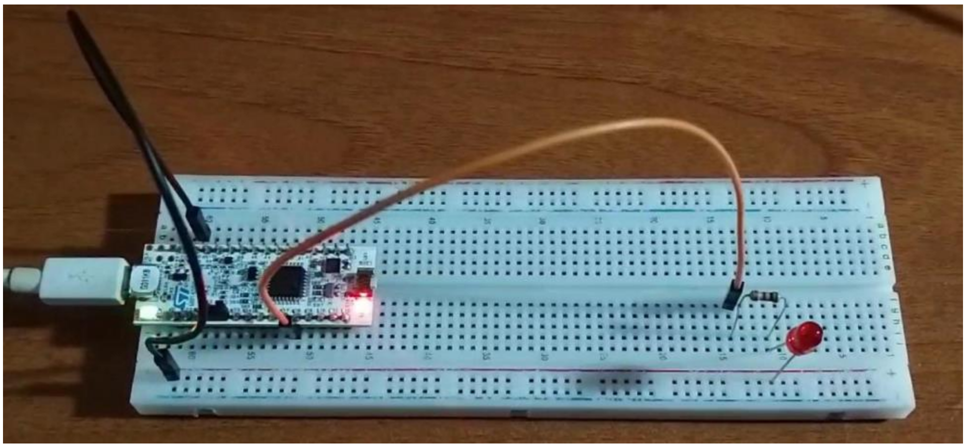

Figure 4 - on the board LED is turn off

2.1.3. Assembly Code and of the Problem I

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
* lab2p1.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 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)          // GPIOB base address
.equ GPIOB_MODER,      (GPIOB_BASE + (0x00)) // GPIOB MODER register offset
.equ GPIOB_ODR,        (GPIOB_BASE + (0x14)) // GPIOB ODR 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:
      /* enable GPIOA clock, bit0 on IOPENR */
      ldr r6, =RCC_IOPENR
      ldr r5, [r6] // R5 = 0x00000000 -> Reset value
      /* movs expects imm8, so this should be fine */
      movs r4, 0x2 // for GPIOB clock enable
      orrs r5, r5, r4
      str r5, [r6] // to send from R5 data to R6 memory
      /* setup for PA8 bit for 16-17 bits in MODER */
      ldr r6, =GPIOB_MODER
      ldr r5, [r6]
      /* cannot do with movs, so use pc relative */
      ldr r4, =0x30000 // to change mode of 8th port
      mvns r4, r4 //inverse
      ands r5, r5, r4 // and ->> inverse + and = bics
      ldr r4, =0x10000
      orrs r5, r5, r4
      str r5, [r6]

loop:
      /*
            set led
            delay
            clear led
      */
      ldr r6, =GPIOB_ODR
      ldr r5, [r6]
      ldr r4, =0x100
      orrs r5, r5, r4
      str r5, [r6]

      ldr r1, =6000000
      bl delay

      ldr r6, =GPIOB_ODR
      ldr r5, [r6]
      ldr r4, =0x100
      bics r5, r5, r4
      str r5, [r6]

      ldr r1, =6000000
      bl delay

      b loop

delay:
      subs r1, #1 // r1 = r1 - 1
      bne delay // r1 is not equal to 0.
      bx lr // r1 is equal to 0.

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

This Assembly code contains a simple block of code that enables the microcontroller to blink an LED. In addition, the 'delay' function allows the written program to wait in the loop for a specified period of time (the desired time is 1 second). 'delay' function decreases the value in r1 by 1 (subs r1, #1). Then, it checks whether r1 is equal to zero with the ‘bne’ (branch if not equal) command. If r1 has a non-zero value, it returns to the 'delay' function and the processor continues to reduce and divide again in the next cycle. If r1 is equal to zero, the function is exited with the ‘bx lr’ command and returned to the place where the function was called with the help of lr (link register). This method is used to provide a wait for a specified period of time (1 second). By changing this value, the waiting time can be set.

2.2. Problem II

2.2.1. Flow Chart and Schematic Diagram

The flow chart of Problem II is as follows.

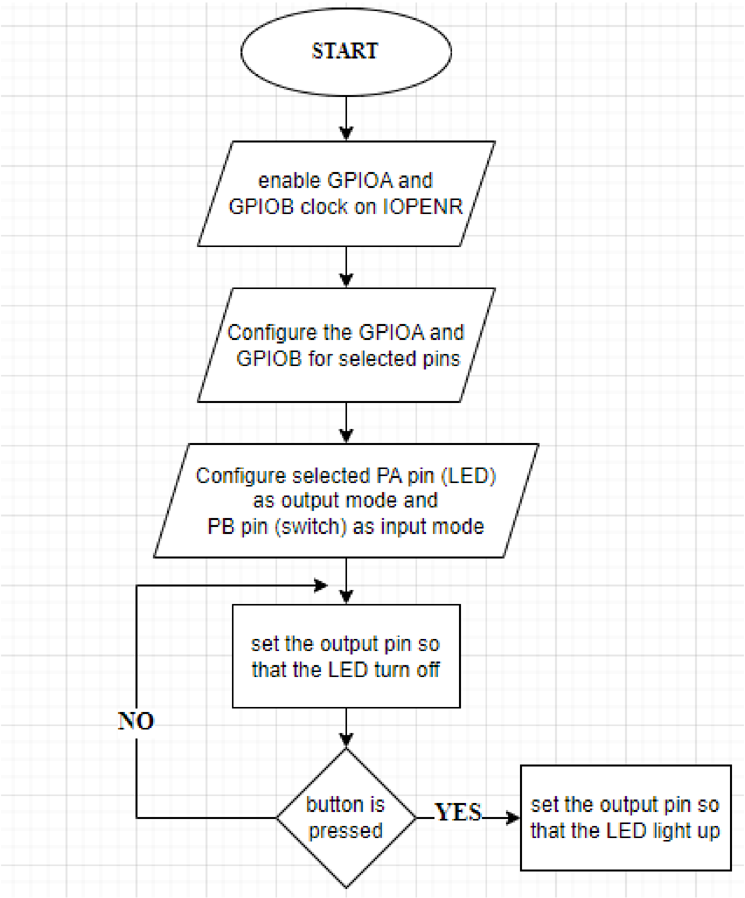
START
enable GPIOA and GPIOB clock on IOPENR
Configure the GPIOA and GPIOB for selected pins
Configure selected PA pin (LED) as output mode and PB pin (switch) as input mode
set the output pin so that the LED turn off
NO
button is pressed
YES
set the output pin so that the LED light up

Figure 5 - flow chart for Problem II

The schematic diagram of Problem II is as follows.

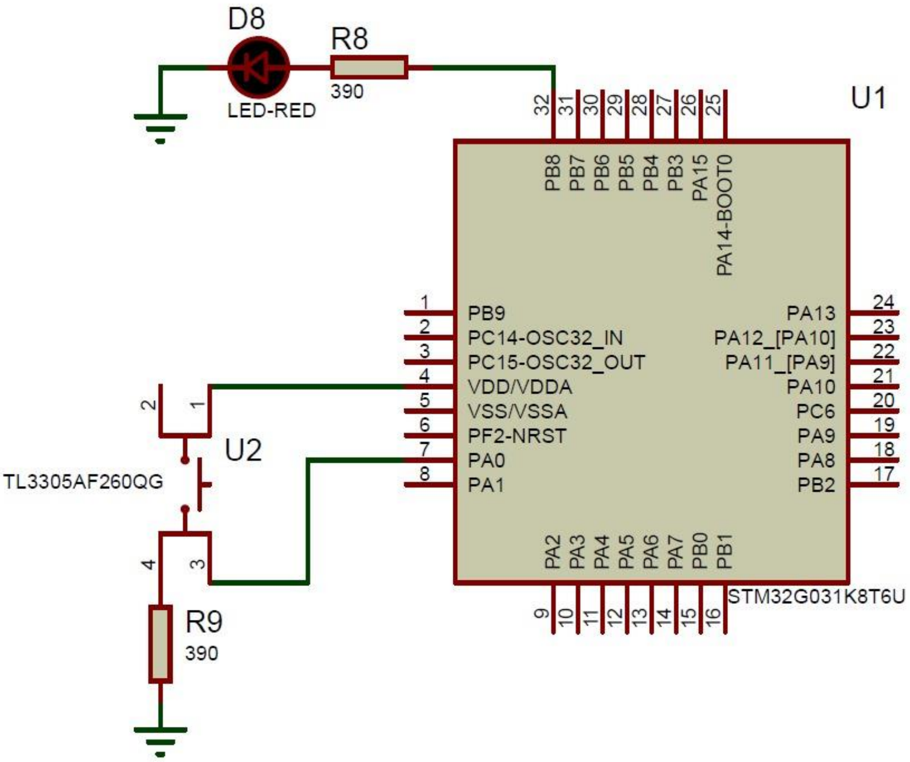

Figure 6 - schematic diagram for Problem II

2.2.2. Installation and Operation of the Circuit

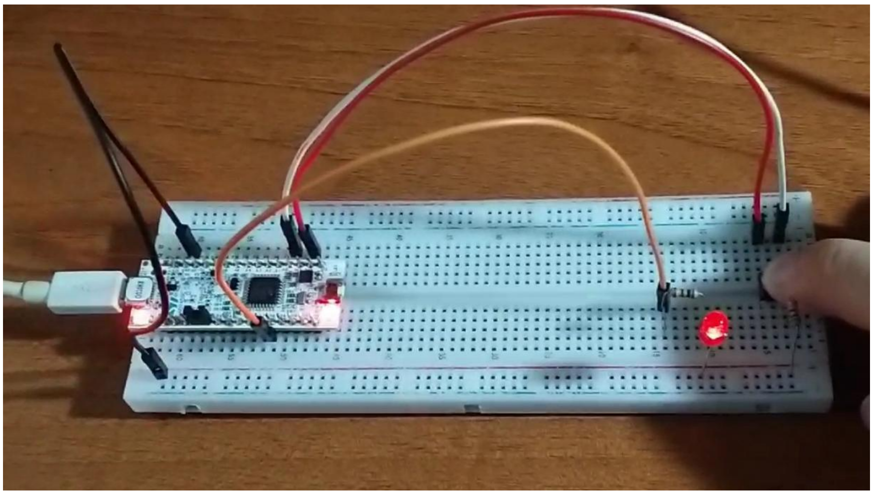

Figure 7 - on board LED turns on when button is pressed

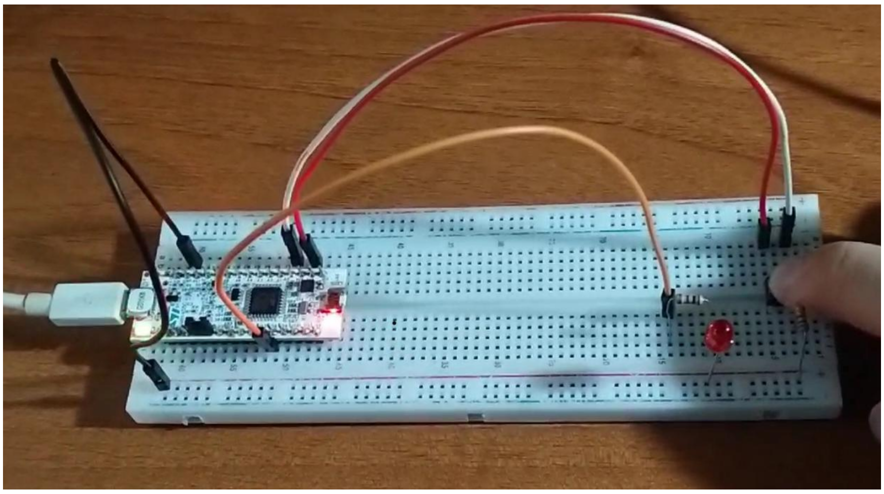

***Figure 8** - LED turns off when button is released*

2.2.3. Assembly Code of the Problem II

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
* lab2p2.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 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 GPIOB_BASE,       (0x50000400)          // GPIOB base address
.equ GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER register offset
.equ GPIOB_MODER,      (GPIOB_BASE + (0x00)) // GPIOB MODER register offset
.equ GPIOA_IDR,        (GPIOA_BASE + (0x10)) // GPIOA IDR register offset
.equ GPIOB_ODR,        (GPIOB_BASE + (0x14)) // GPIOB ODR 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:
      /* enable GPIOA and GPIOB clock on IOPENR */
      ldr r1, =RCC_IOPENR
      ldr r2, [r1]
      // Configure the GPIOA and GPIOB
      movs r4, 0x3
      orrs r2, r2, r4
      str r2, [r1]

      // Configure selected PA0 pin as input mode in MODER
      ldr r1, =GPIOA_MODER
      ldr r2, [r1]
      ldr r4, =0x3
      bics r2, r2, r4
      str r2, [r1]
      // Configure selected PB8 pin as output mode in MODER
      ldr r1, =GPIOB_MODER
      ldr r2, [r1]
      ldr r4, =0x30000
      bics r2, r2, r4
      ldr r4, =0x10000
      orrs r2, r2, r4
      str r2, [r1]


    loop:
        ldr r1, =GPIOB_ODR
        ldr r2, [r1]
        ldr r4,= 0x0
        ands r2, r2, r4
        str r2, [r1]
        ldr r0,=GPIOA_IDR
        ldr r1,[r0]
        ldr r5, =0x1
        ands r1, r1, r5
        ldr r2,=0x1
        cmp r1,r2
        beq button
        b  loop

      button:
        ldr r1, =GPIOB_ODR
        ldr r2, [r1]
        ldr r4,= 0x100
        orrs r2, r2, r4
        str r2, [r1]
        b  loop
      /* for(;;); */
      b .

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

In Problem 2, the LED is turned on and off with the button control. As can be seen from Figure 6, PA_0 pin is used for input and PB_8 pin is used for output. GPIOA and GPIOB are configured as in the Assembly code above, PA0 is set as the input mode. PB8 pin is set as output mode. Subsequently, the 9th bit of ODR is reset to turn off the LED within the loop. Then, the input data from the button in the loop is checked. When the button is pressed, the LED turns on with the 'button' function.

2.3. Problem III

2.3.1. Flow Chart and Schematic Diagram

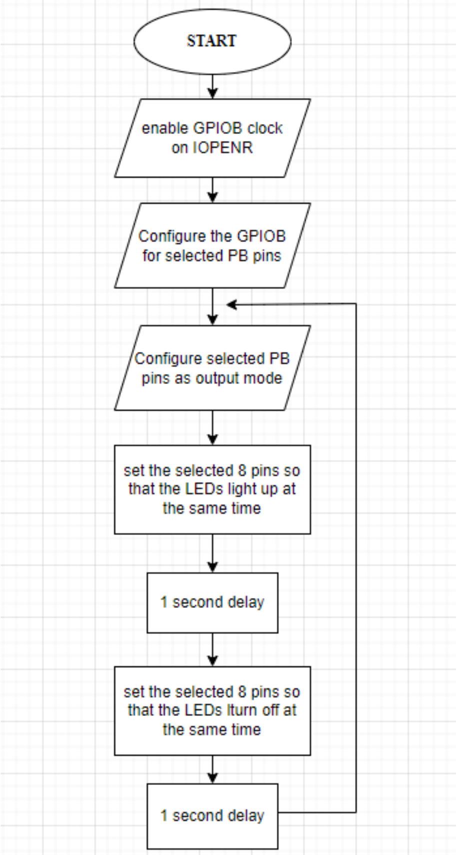
START
enable GPIOB clock on IOPENR
Configure the GPIOB for selected PB pins
Configure selected PB pins as output mode
set the selected 8 pins so that the LEDs light up at the same time
1 second delay
set the selected 8 pins so that the LEDs lturn off at the same time
1 second delay

Figure 9 - flow chart for Problem III

The schematic diagram of Problem III is as follows.

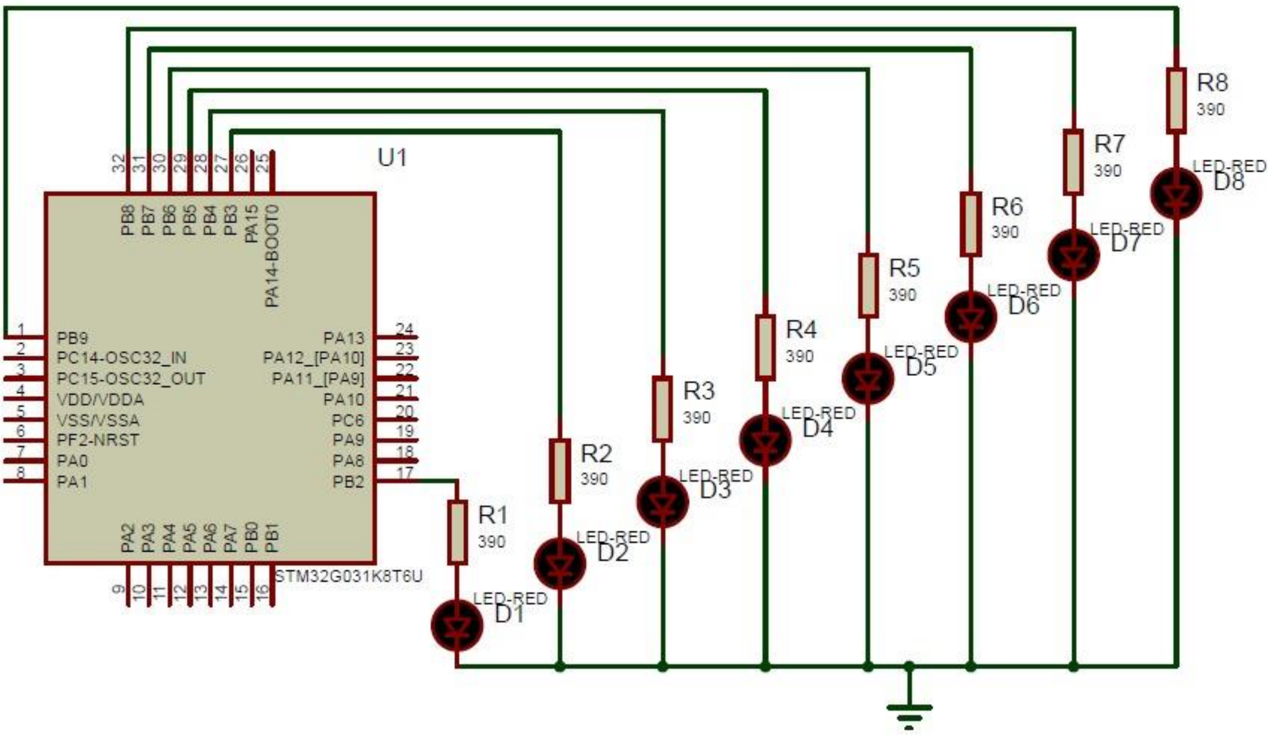

Figure 10 – schematic diagram for Problem III

2.3.2. Installation and Operation of the Circuit

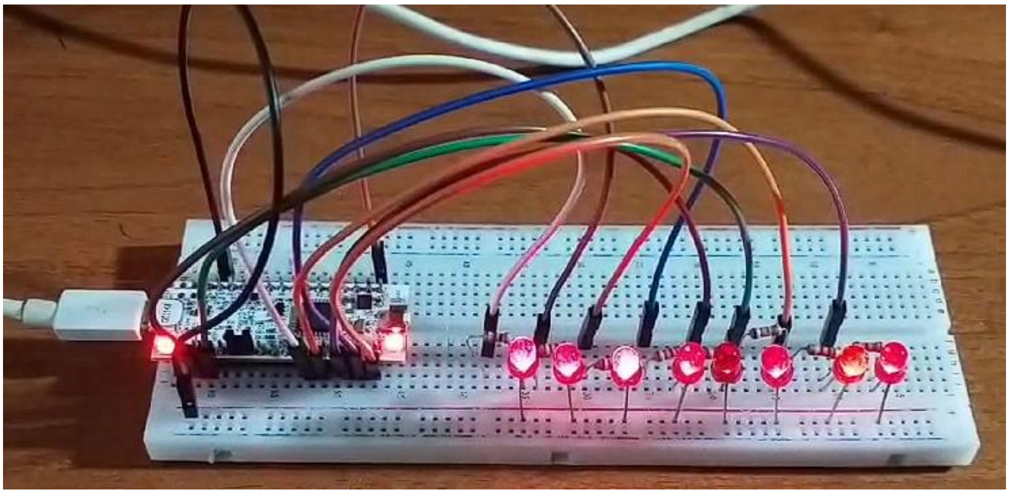

Figure 11 – all LEDs turn on at the same time

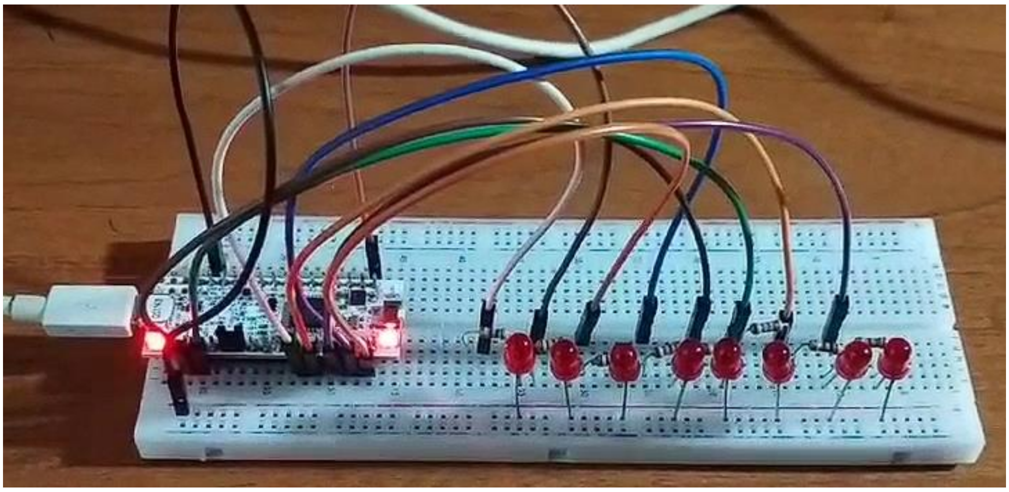

Figure 12 - all LEDs turn off at the same time after a 1 second delay

2.3.3. Assembly Code of the Problem III

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
* lab2p3.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 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)          // GPIOB base address
.equ GPIOB_MODER,      (GPIOB_BASE + (0x00)) // GPIOB MODER register offset
.equ GPIOB_ODR,        (GPIOB_BASE + (0x14)) // GPIOB ODR 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:
     /* enable GPIOB clock, bit2 on IOPENR */
     ldr r1, =RCC_IOPENR
     ldr r2, [r1]
     // Configure the GPIOB for PB pins
     movs r4, 0x2
     orrs r2, r2, r4
     str r2, [r1]
     // Configure selected PB pins as output mode in MODER
     ldr r1, =GPIOB_MODER
     ldr r2, [r1]
     ldr r4, =0xFFFF0
     bics r2, r2, r4
     ldr r4, =0x55550
     orrs r2, r2, r4
     str r2, [r1]
```

```
loop:
    /* turn on leds in ODR */
    ldr r6, =GPIOB_ODR
    ldr r5, [r6]
    ldr r4, =0x3FC
    orrs r5, r5, r4
    str r5, [r6]

    ldr r1, =6000000 // ~ rate of 1 second
    bl delay

    /* turn off leds in ODR */
    ldr r6, =GPIOB_ODR
    ldr r5, [r6]
    ldr r4, =0x3FC
    bics r5, r5, r4 // r5 = r5 & ~r4
    str r5, [r6]

    ldr r1, =6000000 // ~ rate of 1 second
    bl delay
    b loop

delay:
    subs r1, #1 // r1 = r1 - 1
    bne delay // r1 is not equal to 0.
    bx lr // r1 is equal to 0.

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

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

Firstly, in problem 3, 8 external LEDs are connected to the board and all LEDs are toggled at the same time at a rate of 1 second. As can be seen from Figure 10; PB_2, PB_3, PB_4, PB_5, PB_6, PB_7, PB_8 and PB_9 pins is used for output (for LEDs). GPIOB is configured as in the Assembly code above, selected pins are set as output mode. All LEDs are turned on in ODR with basic logic operations in the loop at the same time. Then, with the help of the 'delay' function, all LEDs toggle in 'loop' at the same time at a rate of 1 second.

2.4. Problem IV

2.4.1. Flow Chart and Schematic Diagram

The flow chart of Problem IV is as follows.

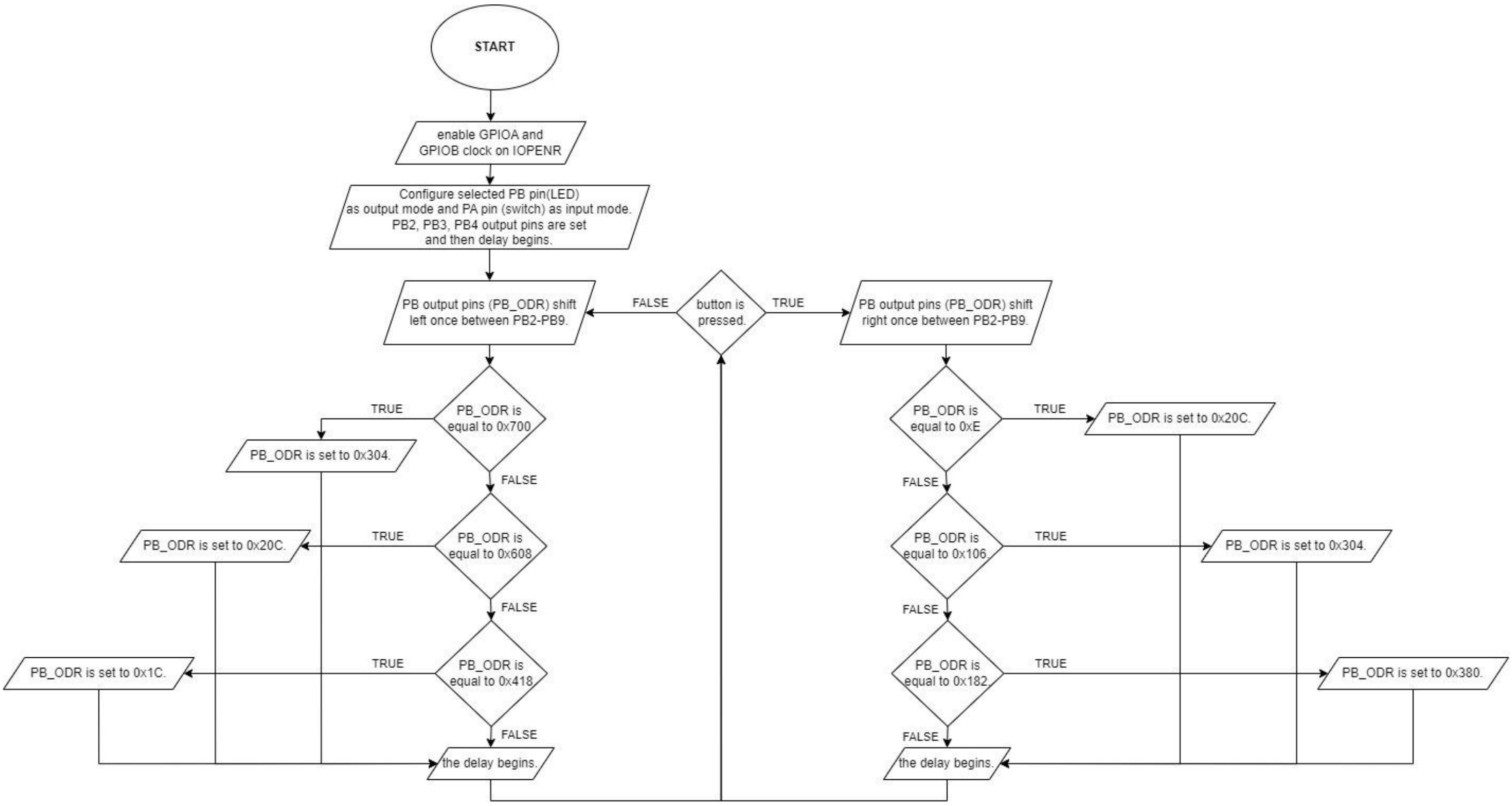

Figure 13 - flow chart for Problem IV

The schematic diagram of Problem IV is as follows.

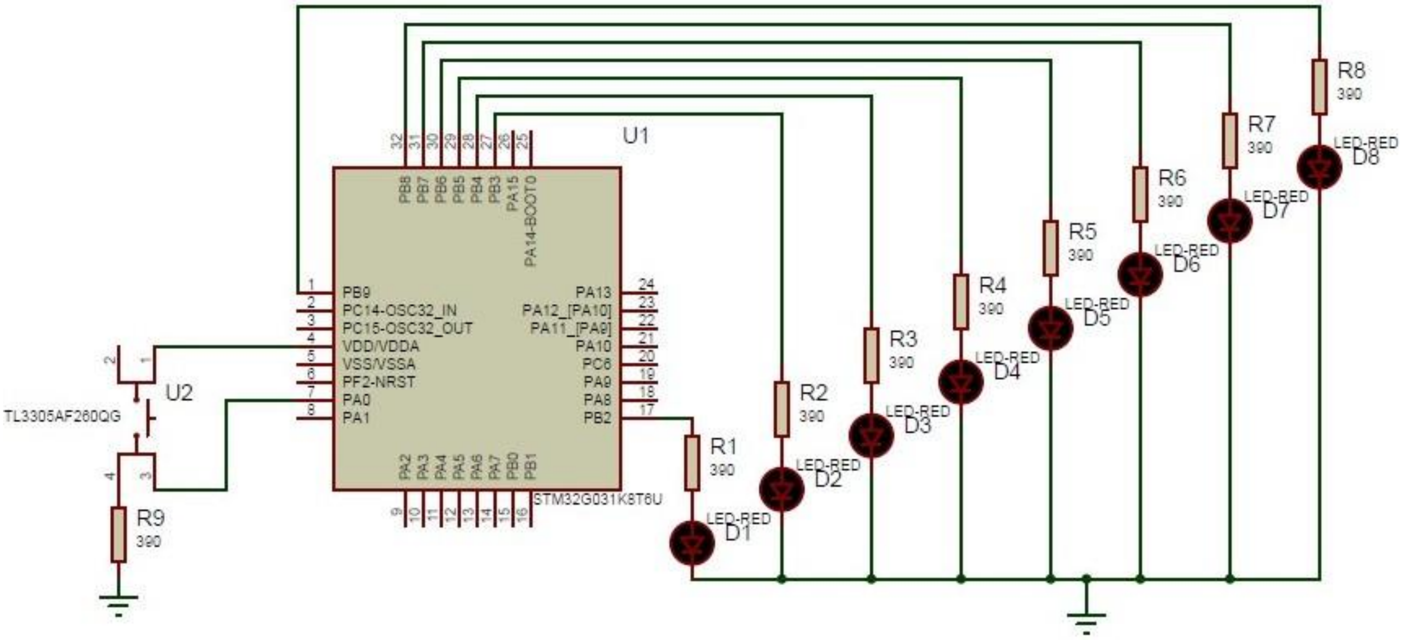

Figure 14 – schematic diagram for Problem IV

2.4.2. Installation and Operation of the Circuit

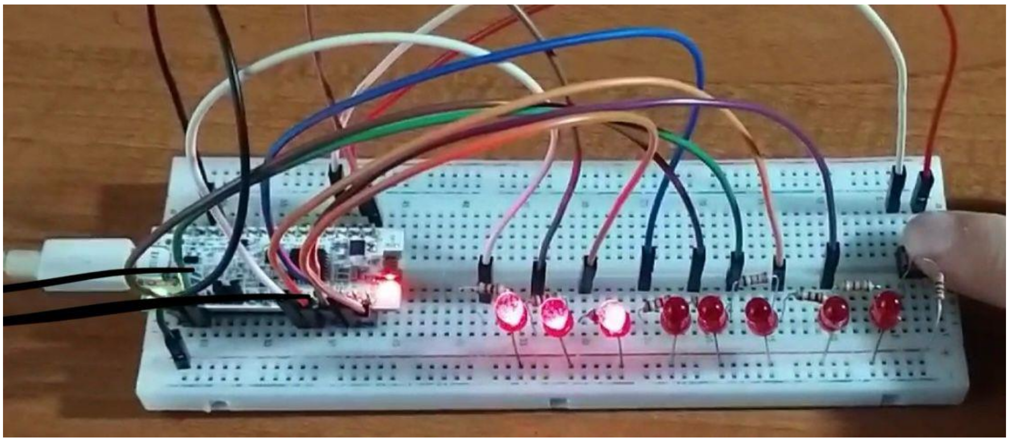

Figure 15

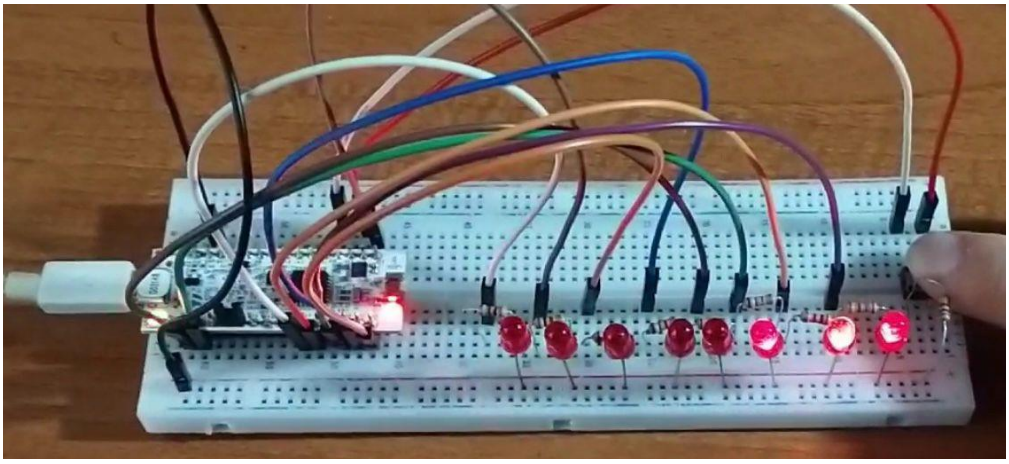

Figure 16

2.4.3. Assembly Code of the Problem IV

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
* lab2p4.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 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 GPIOB_BASE,       (0x50000400)          // GPIOB base address
.equ GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER register offset
.equ GPIOB_MODER,      (GPIOB_BASE + (0x00)) // GPIOB MODER register offset
.equ GPIOA_IDR,            (GPIOA_BASE + (0x10)) // GPIOA IDR register offset
.equ GPIOB_ODR,        (GPIOB_BASE + (0x14)) // GPIOB ODR 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:
      /* enable GPIOA clock, bit2 on IOPENR */
    ldr r6, =RCC_IOPENR
    ldr r2, [r6]
   // Configure the GPIOA for PA pins
    movs r4, 0x3
    orrs r2, r2, r4
    str r2, [r6]


    ldr r0, =GPIOA_MODER
    ldr r2, [r0]
    ldr r4, =0x3
    bics r2, r2, r4
    str r2, [r0]

    ldr r1, =GPIOB_MODER
    ldr r2, [r1]
    ldr r4, =0xFFFF0
    bics r2, r2, r4
    ldr r4, =0x55550
      orrs r2, r2, r4
    str r2, [r1]
      ldr r1, =GPIOB_ODR
    ldr r2,= 0x1c
    str r2, [r1]
    ldr r3, =600000
      bl delay
      ldr r4, =GPIOA_IDR
      ldr r0, [r4]
```

```
leftDirect:
        ldr r2, [r1]
            lsls r2, #1
            ldr r4, =0x700
            cmp r2,r4
            beq t1
            ldr r4, =0x608
            str r2, [r1]
            cmp r2,r4
            beq t2
            ldr r4, =0x418
            cmp r2,r4
            beq t3
            str r2, [r1]
            ldr r3, =600000
            bl delay
            bl button1
            b leftDirect
t1:
            ldr r2,= 0x304
          str r2, [r1]
          ldr r3, =600000
          bl delay
          bl button1
          b leftDirect
t2:
          ldr r2,= 0x20c
          str r2, [r1]
          ldr r3, =600000
          bl delay
          bl button1
          b leftDirect
t3:
            ldr r2,= 0x1c
      str r2, [r1]
      ldr r3, =600000
          bl delay
          bl button1
          b leftDirect
RightDirect:
        ldr r2, [r1]
            lsrs r2, #1
            ldr r4, =0xe
            cmp r2,r4
            beq t4
            ldr r4, =0x106
            str r2, [r1]
            cmp r2,r4
            beq t5
            ldr r4, =0x182
            cmp r2,r4
            beq t6
            str r2, [r1]
            ldr r3, =600000
            bl delay
            bl button2
            b  RightDirect
```

```
t4:
            ldr r2,= 0x20c
          str r2, [r1]
          ldr r3, =600000
          bl delay
          bl button2
          b  RightDirect
t5:
          ldr r2,= 0x304
          str r2, [r1]
          ldr r3, =600000
          bl delay
          bl button2
          b  RightDirect
t6:
            ldr r2,= 0x380
      str r2, [r1]
      ldr r3, =600000
          bl delay
          bl button2
          b  RightDirect

button1:
        ldr r4, =GPIOA_IDR
        ldr r5, [r4]
        cmp r0 , r5
            bne RightDirect
            bx lr

button2:
            ldr r4, =GPIOA_IDR
        ldr r5, [r4]
        cmp r0 , r5
            beq leftDirect
            bx lr
delay:
      subs r3, #1 // r1 = r1 - 1
            bne delay // r1 is not equal to 0.
            bx lr // r1 is equal to 0.
      /* for(;;); */

b .
```

In the Main section, GPIOA and GPIOB are first configured with the RCC_IOPENR command. The pin (PA_0) to which the button is connected is set to 00 to operate in input mode. Then, each pin of GPIOB_MODER, from PB_2 to PB9, to which the LEDs are connected, is set to 01, that is, output mode. Using GPIOB_ODR, LEDs 1, 2 and 3 are powered and a delay is applied. Inside the 'leftDirect' function, the powered LEDs are shifted to the left once and a certain delay is applied each time. While the powered LEDs are shifted to the left, t1, t2 and t3 functions are used to reset the LEDs to the beginning when it is the turn of the last 3 LEDs. The overflow process is controlled with the CMP command. Here, the 'button1' function is used to check whether the button is pressed after each scrolling operation. The 'button1' function is made by comparing the current input entries with the initial input entries kept in the main section. The same operations are carried out with the 'RightDirect' function, and when any button is pressed, the LEDs that light up start to shift to the opposite side.

2.5. Problem V

2.5.1. Flow Chart and Schematic Diagram

The flow chart of Problem V is as follows.

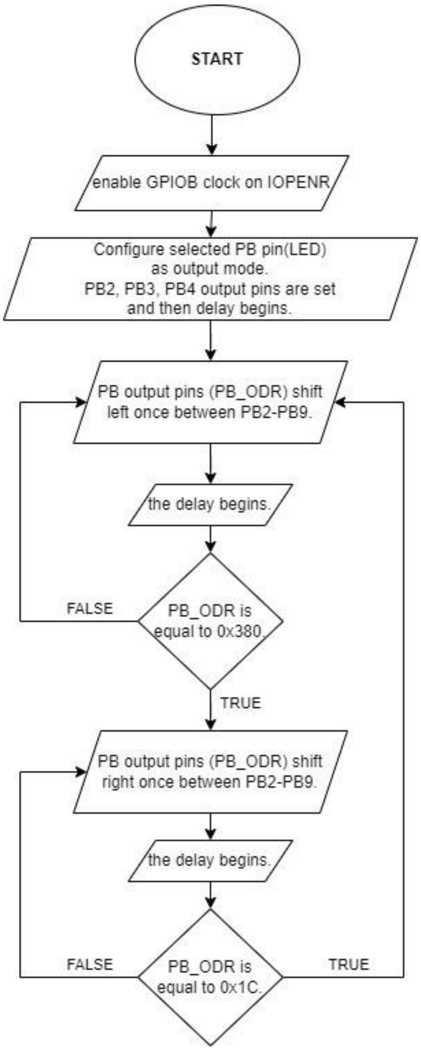
START
enable GPIOB clock on IOPENR
Configure selected PB pin(LED) as output mode. PB2, PB3, PB4 output pins are set and then delay begins.
PB output pins (PB_ODR) shift left once between PB2-PB9.
the delay begins.
PB_ODR is equal to 0x380.
FALSE
TRUE
PB output pins (PB_ODR) shift right once between PB2-PB9.
the delay begins.
PB_ODR is equal to 0x1C.
FALSE
TRUE

Figure 16 - flow chart for Problem V

The schematic diagram of Problem V is as follows.

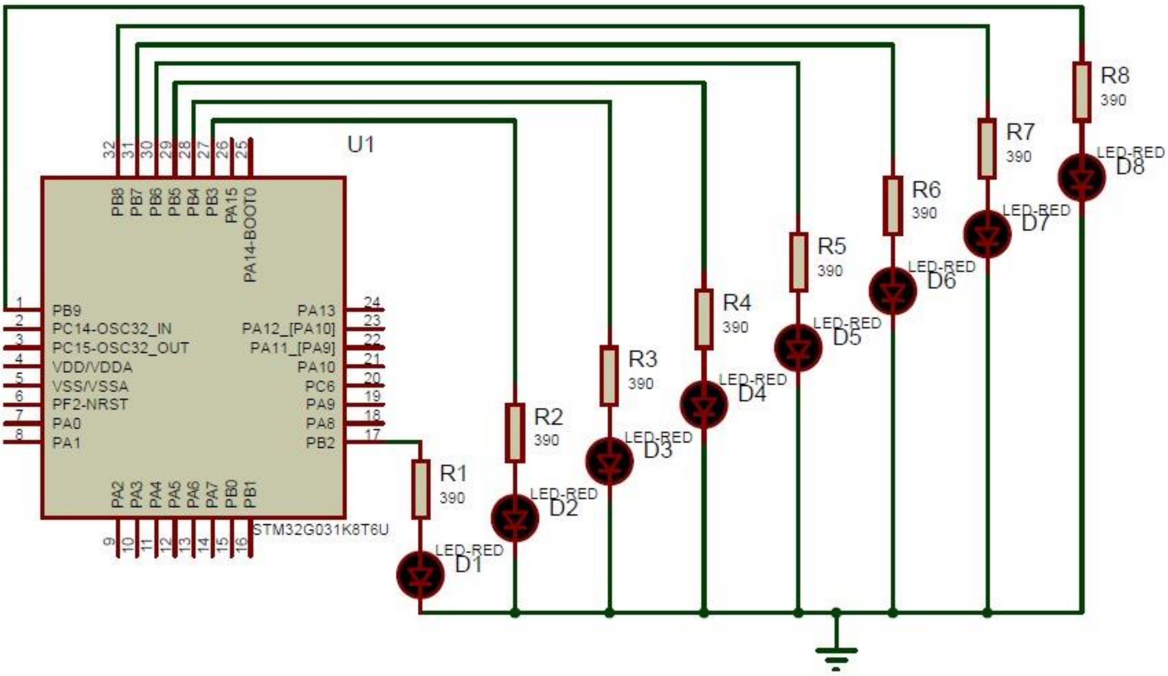

Figure 17 – schematic diagram for Problem V

2.5.2. Installation and Operation of the Circuit

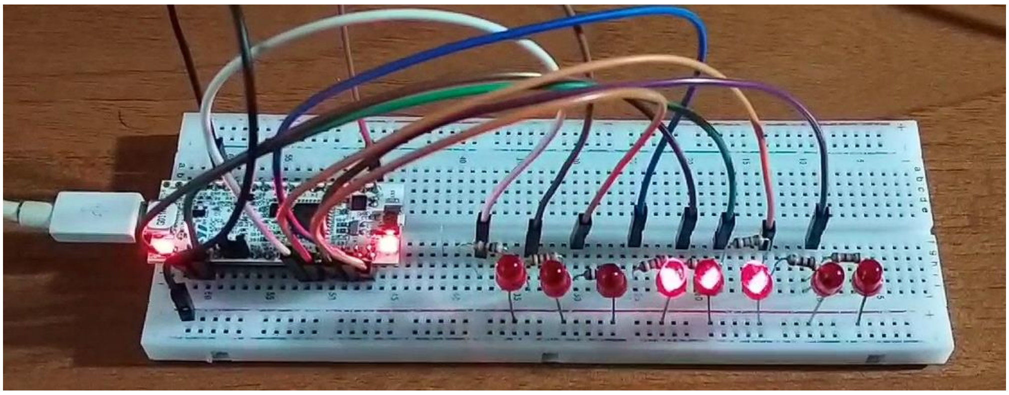

Figure 18 - for Problem V, the image of the circuit at any time

2.5.3. Assembly Code of the Problem V

```
/* author: Umut Mehmet Erdem | Arda Derici | Serdar Başyemenici
* lab2p5.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 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)          // GPIOB base address
.equ GPIOB_MODER,      (GPIOB_BASE + (0x00)) // GPIOB MODER register offset
.equ GPIOB_ODR,        (GPIOB_BASE + (0x14)) // GPIOB ODR 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:
      /* enable GPIOB clock on IOPENR */
    ldr r6, =RCC_IOPENR
    ldr r2, [r6]
   // Configure the GPIOB for PB pins
    movs r4, 0x2
    orrs r2, r2, r4
    str r2, [r6]

    ldr r1, =GPIOB_MODER
    ldr r2, [r1]
    ldr r4, =0xFFFF0
    bics r2, r2, r4
    ldr r4, =0x55550
      orrs r2, r2, r4
    str r2, [r1]
      ldr r1, =GPIOB_ODR
    ldr r2,= 0x1c
    str r2, [r1]
    ldr r3, =600000
      bl delay

leftDirect:
        ldr r2, [r1]
            lsls r2, #1
            ldr r4, =0x380
            str r2, [r1]
            ldr r3, =600000
            bl delay
            cmp r2, r4
            beq RightDirect
            b  leftDirect

RightDirect:
        ldr r2, [r1]
            lsrs r2, #1
            ldr r4, =0x1c
            str r2, [r1]
            ldr r3, =600000
            bl delay
            cmp r2,r4
            beq leftDirect
            b  RightDirect

delay:
      subs r3, #1 // r1 = r1 - 1
            bne delay // r1 is not equal to 0.
            bx lr // r1 is equal to 0.
      /* for(;;); */

b .
```

In the Main section, GPIOB is first configured with the RCC_IOPENR command. GPIOB_MODER, from PB2 to PB9, to which the LEDs are connected, is set so that each pin is 01, that is, in output mode. Using GPIOB_ODR, LEDs 1, 2 and 3 are powered and a delay is applied.. With the 'leftDirect' command, the powered pins are shifted to the left and a certain delay is applied. While the powered LEDs are shifted to the left, when it came to the last 3 LEDs, the power is directed to the 'RightDirect' function to be in the opposite direction, this process is carried out with the 'CMP' and 'BEQ' commands. Likewise, when it comes to the last 3 LEDs again, the 'leftDirect' function is directed to reverse the scrolling process again.

3. Conclusions

In this ELEC 335 laboratory 2 experiment, working with subroutines and functions was learned. For this purpose, functions were transitioned from branch to branch by controlling the b (branch) command with certain conditions (eq, neq, exc.) with commands such as the CMP comparison command. In this way, LED control can be made with a button or delay in laboratory 2 experiment. Information about activating the pin and setting the pin to input-output mode has been realized in practice with Assembly.

4. References

[1] https://github.com/fcayci/stm32g0

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

[3] https://www.st.com/resource/en/datasheet/stm32g031k8.pdf

[4] https://www.st.com/resource/en/schematic_pack/mb1455-g031k8-c01_schematic.pdf

[5] https://www.st.com/resource/en/user_manual/um2591-stm32g0-nucleo32-board-mb1455-stmicroelectronics.pdf

[6] drawio.com