GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC335 – MICROPROCESSORS LABAORATORY

LAB 1

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

PROBLEMS

Problem 1

- Identify all the components and explain their usage

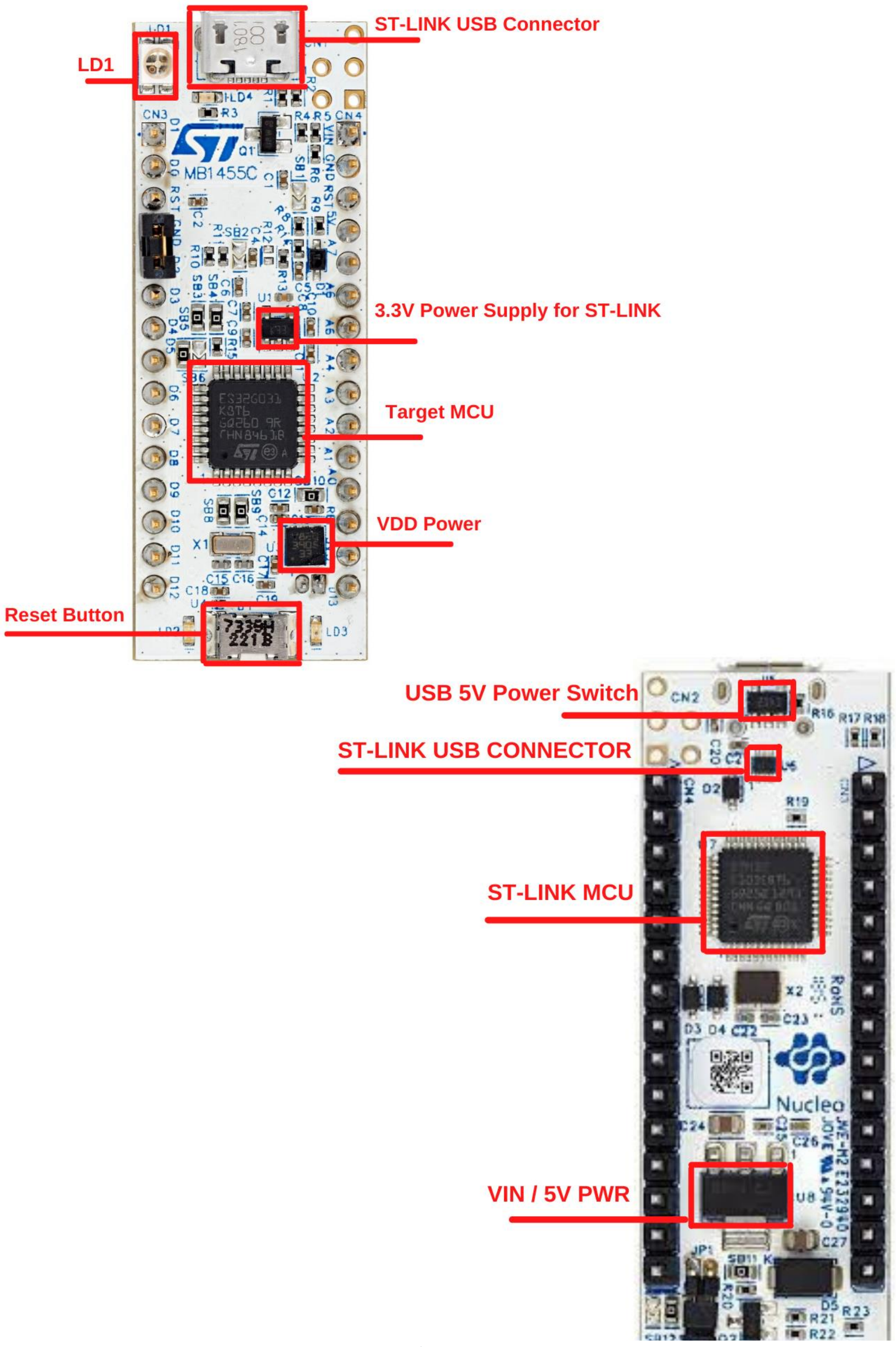
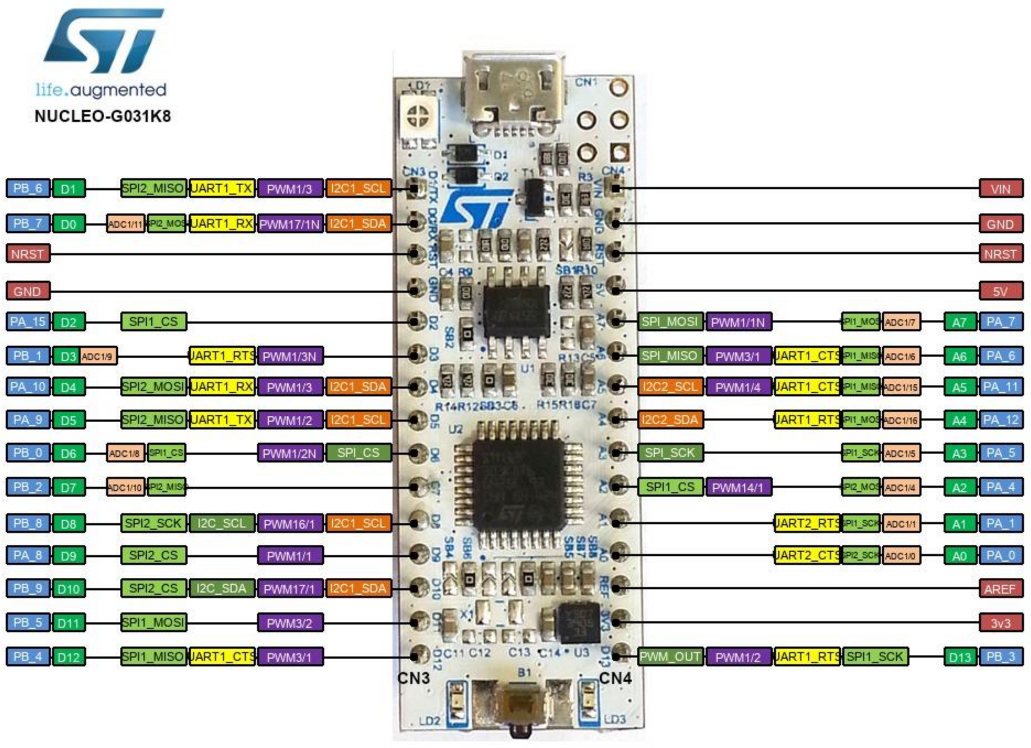


Figure 1: NucleoG031K8 Layout

- **3.3V Power Supply for ST-LINK →** USB üzerinden verilen besleme kart üzerindeki sabit regülatör tarafından 5V’a düşürülmektedir. ST-LINK için güç kaynağı görevi gören bu regülatör ise 5V’u 3V’a düşürerek ST-LINK’e güç vermektedir.
- **ST-LINK MCU →** ST-LINK MCU allows the desired code to be uploaded to the target MCU via UART. When it is desired to load code via CubeIDE, it is first connected to the ST-LINK MCU. ST-LINK MCU deletes the code in the memory of the target MCU and loads the desired code.
- **VDD Power →** It is the main regulator that reduces the voltage supplied via USB to 5V. It is the supply of most components on the board.
- **Reset Button →** It is the button that restarts the code running on the target MCU from the beginning. It makes the RST pin 1 on the target MCU when pressed.
- **USB 5V Power Switch →** It is the regulator that goes by the name of Step-Down or Buck Switching regulator. In case of large voltage values, it offers a more reasonable voltage range as an output.
- **ST-LINK USB CONNECTOR →** It is the micro USB input used to load the code to the target MCU or to feed the card.
- **Target MCU →** It is the MCU that does what is requested in the loaded code. It controls the pin outputs as desired in the code, and saves the loaded code to the ROM inside. In this way, every time the card is powered, it constantly fulfills what is required in the code.

- Explain all the connected peripherals and their pin connections with the microcontroller.



life.augmented
NUCLEO-G031K8
PB_6 D1 SPI2_MISO UART1_TX PWM1/3 I2C1_SCL
PB_7 D0 ADC1/11 SPI2_MOSI UART1_RX PWM17/1N I2C1_SDA
NRST
GND
PA_15 D2 SPI1_CS
PB_1 D3 ADC1/9 UART1_RTS PWM1/3N
PA_10 D4 SPI2_MOSI UART1_RX PWM1/3 I2C1_SDA
PA_9 D5 SPI2_MISO UART1_TX PWM1/2 I2C1_SCL
PB_0 D6 ADC1/8 SPI1_CS PWM1/2N SPI_CS
PB_2 D7 ADC1/10 SPI2_MISO
PB_8 D8 SPI2_SCK I2C_SCL PWM16/1 I2C1_SCL
PA_8 D9 SPI2_CS PWM1/1
PB_9 D10 SPI2_CS I2C_SDA PWM17/1 I2C1_SDA
PB_5 D11 SPI1_MOSI PWM3/2
PB_4 D12 SPI1_MISO UART1_CTS PWM3/1
CN3
CN4
VIN
GND
NRST
5V
SPI_MOSI PWM1/1N SPI1_MOSI ADC1/7 A7 PA_7
SPI_MISO PWM3/1 UART1_CTS SPI1_MISO ADC1/6 A6 PA_6
I2C2_SCL PWM1/4 UART1_CTS SPI1_MISO ADC1/15 A5 PA_11
I2C2_SDA UART1_RTS SPI1_MOSI ADC1/16 A4 PA_12
SPI_SCK SPI1_SCK ADC1/5 A3 PA_5
SPI1_CS PWM14/1 SPI2_MOSI ADC1/4 A2 PA_4
UART2_RTS SPI1_SCK ADC1/1 A1 PA_1
UART2_CTS SPI2_SCK ADC1/0 A0 PA_0
AREF
3v3
PWM_OUT PWM1/2 UART1_RTS SPI1_SCK D13 PB_3
LD2
LD3
B1

Figure 2: NucleoG031K8 Pinout

Problem 2

- In this problem you are asked to write code that will light up the on-board LED connected to pin PC6.

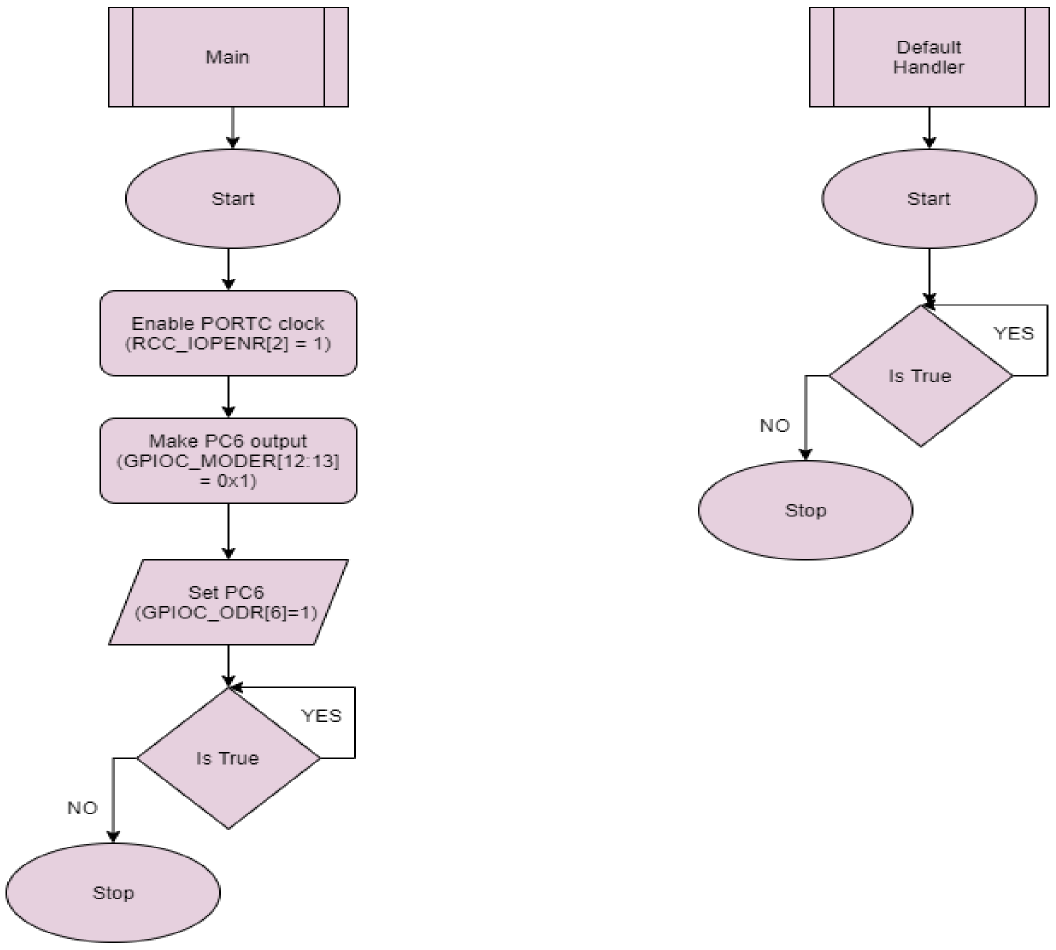


Figure 3: Problem 2 Flowchart

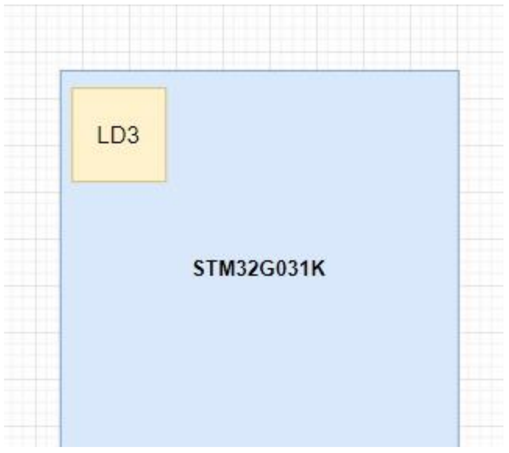


Figure 4: Problem 2 Block Diagram

```
/*

 * asm.s

 *

 * author: 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 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 GPIOC_BASE,       (0x50000800)          // GPIOC base address

.equ GPIOC_MODER,      (GPIOC_BASE + (0x00)) // GPIOC MODER
register offset
```

```
.equ GPIOC_ODR,        (GPIOC_BASE + (0x14)) // GPIOC 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 GPIOC clock, bit2 on IOPENR */
	ldr r6, =RCC_IOPENR
	ldr r5, [r6]
	/* movs expects imm8, so this should be fine */
	movs r4, 0x4
	orrs r5, r5, r4
	str r5, [r6]
	/* setup PC6 for led 01 for bits 12-13 in MODER */
	ldr r6, =GPIOC_MODER
	ldr r5, [r6]
	/* cannot do with movs, so use pc relative */
	ldr r4, =0x3000
	mvns r4, r4
	ands r5, r5, r4
	ldr r4, =0x1000
	orrs r5, r5, r4
	str r5, [r6]
	/* turn on led connected to C6 in ODR */
```

```
ldr r6, =GPIOC_ODR

ldr r5, [r6]

movs r4, 0x40

orrs r5, r5, r4

str r5, [r6]

/* for(;;); */

b .

/* this should never get executed */

nop
```

- Using a voltmeter, measure the voltage on the PC6 pin when the LED is on, and when the LED is off. Explain and justify the results.

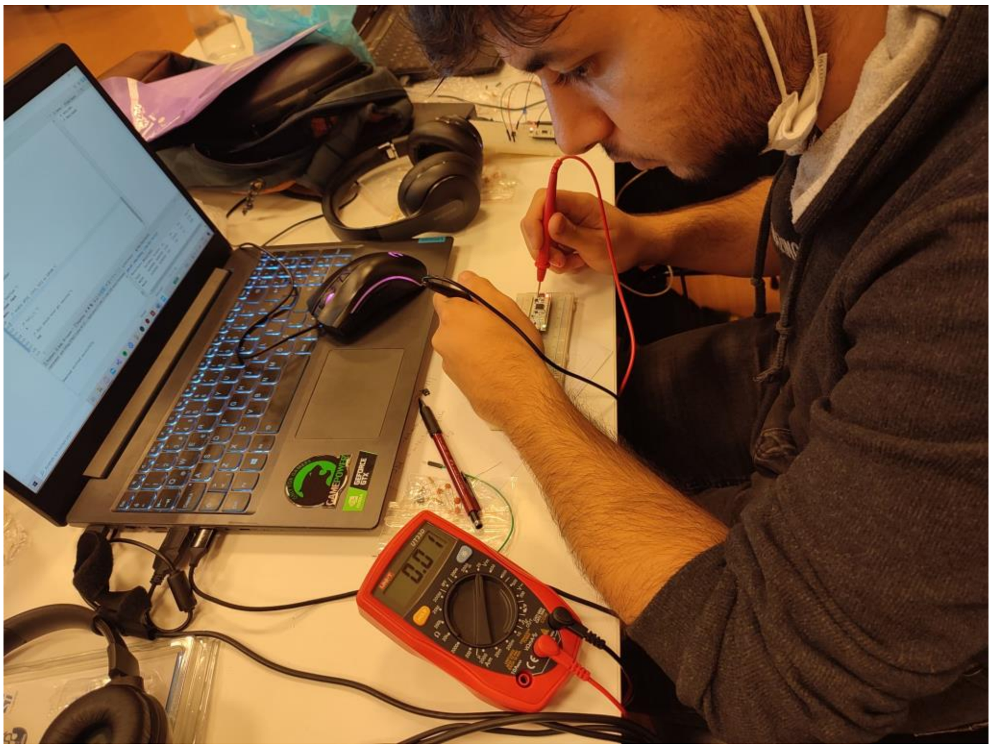


Figure 5: Led is OFF

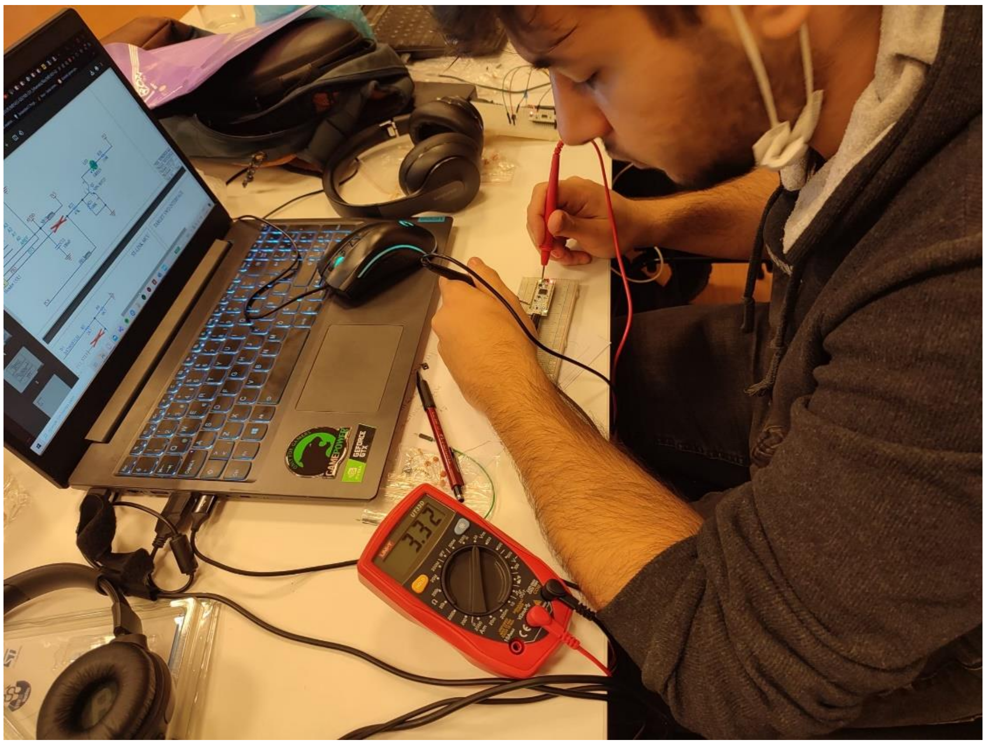


Figure 6: Led is ON

The Led3 is on with the help of the pc6 pin. In order to burn this led, the pc6 pin must be made high. When this pin is connected, the current goes to the base of the Q2 transistor and enables the transistor to be turned on. When this transistor is turned on, led3 becomes short with ground and the led turns on. The pc6 pin must be in the low state when the led is off, so the voltage isn't visible when measuring the voltage of the pc6 pin.

- Find out the nominal voltage and absolute maximum voltage values of your microcontroller pins.

VDD	3.31V
VIN	0.01V
VREF	3.32V

Problem 3

- In this problem you are asked to write code that will light up 4 external LEDs connected to the board.

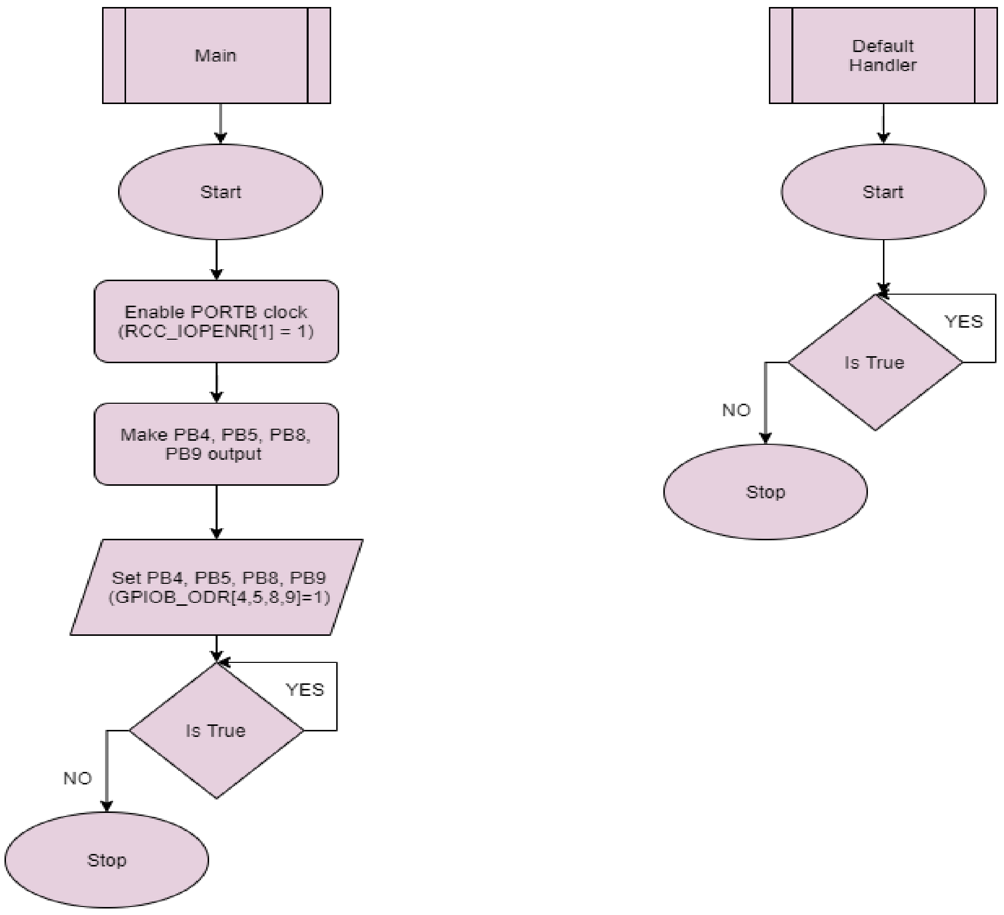

Main
Start
Enable PORTB clock (RCC_IOPENR[1] = 1)
Make PB4, PB5, PB8, PB9 output
Set PB4, PB5, PB8, PB9 (GPIOB_ODR[4,5,8,9]=1)
Is True
YES
NO
Stop
Default Handler
Start
Is True
YES
NO
Stop

Figure 7: Problem 3 Flowchart

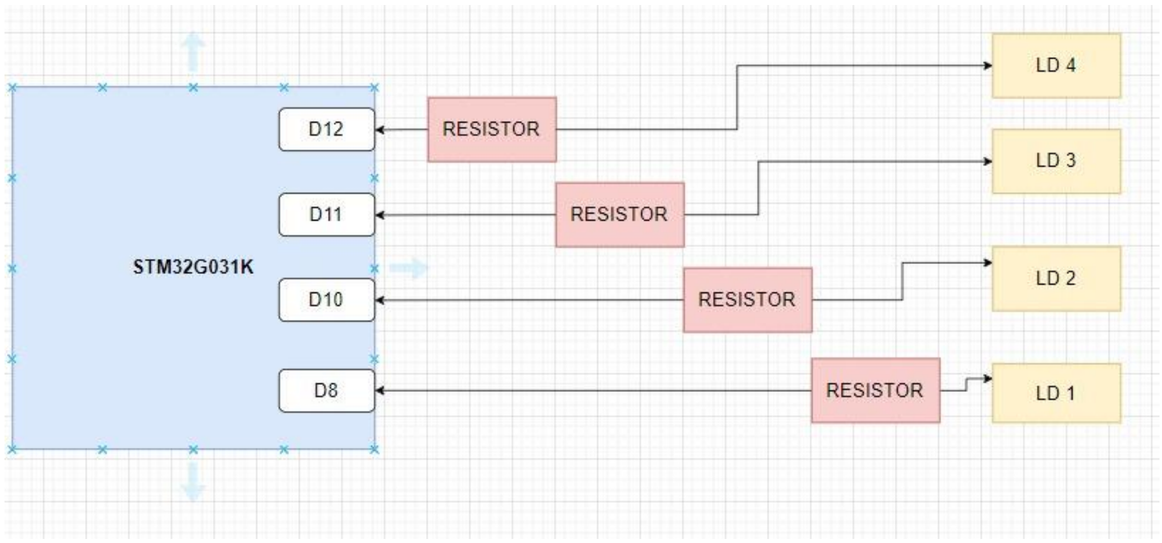


Figure 8: Problem 3 Block Diagram

```
/*

 * asm.s

 *

 * authors: Emirhan KÖSE, RUVEYDA DİLARA GUNAL, 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


/* 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 GPIOC clock, bit2 on IOPENR */

     ldr r6, =RCC_IOPENR

     ldr r5, [r6]

     /* r4 --> 0010   PORT B is enable*/

     movs r4, 0x2

     orrs r5, r5, r4

     str r5, [r6]


     /*mode 4 mode 5 mode 8 and mode 9 are adjusted as output 01
*/

     ldr r6, =GPIOB_MODER

     ldr r5, [r6]

     /* cannot do with movs, so use pc relative */

     ldr r4, =0xE0E00

     mvns r4, r4

     ands r5, r5, r4

     ldr r4, =0x50500

     orrs r5, r5, r4

     str r5, [r6]


     /pin 4 5 8 and 9 are output high/

     ldr r6, =GPIOB_ODR

     ldr r5, [r6]

     ldr r4,=0x330

     orrs r5, r5, r4

     str r5, [r6]
```

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

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

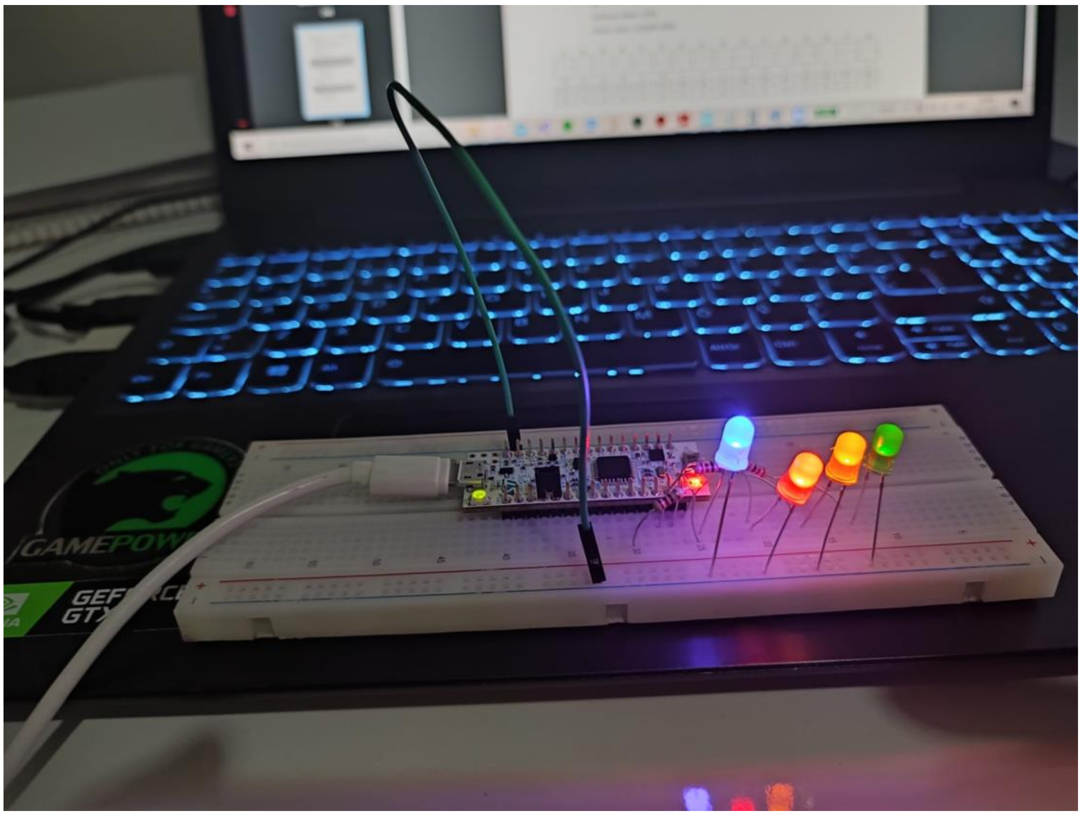


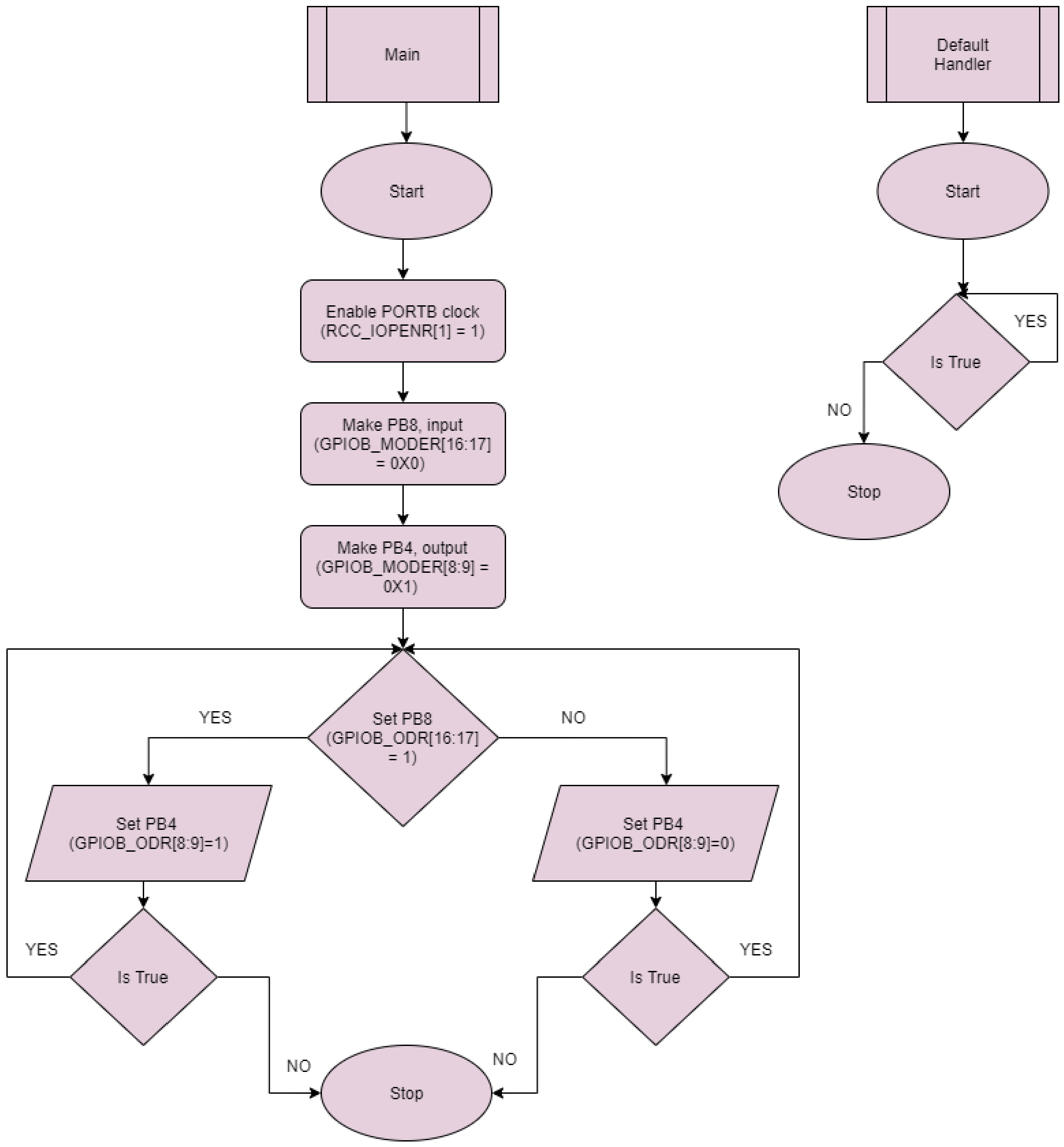
Figure 9: Four leds toggle circuit

- Power off your board, then power back on. (Disconnect the cable) Are the LEDs light up again? Explain and justify the results.

  The codes are written to the rom (read only memory) via st-link. We can think of Rom as a computer's memory. As the information on the disk is not lost when the computer is turned off, the information on the rom is not lost either. When the power is turned on again, the code is written in it and it continues to work in the same way.

Problem 4

- In this problem you are asked to write code that will light up 1 external LED connected to the board using an external push-button. Whenever the button is pressed, the LED should be on, and whenever the button is released, the LED should be off.



Main
Start
Enable PORTB clock (RCC_IOPENR[1] = 1)
Make PB8, input (GPIOB_MODER[16:17] = 0X0)
Make PB4, output (GPIOB_MODER[8:9] = 0X1)
YES
Set PB8 (GPIOB_ODR[16:17] = 1)
NO
Set PB4 (GPIOB_ODR[8:9]=1)
Set PB4 (GPIOB_ODR[8:9]=0)
YES
Is True
Is True
YES
NO
Stop
NO
Default Handler
Start
YES
Is True
NO
Stop

Figure 10: Problem 4 Flowchart

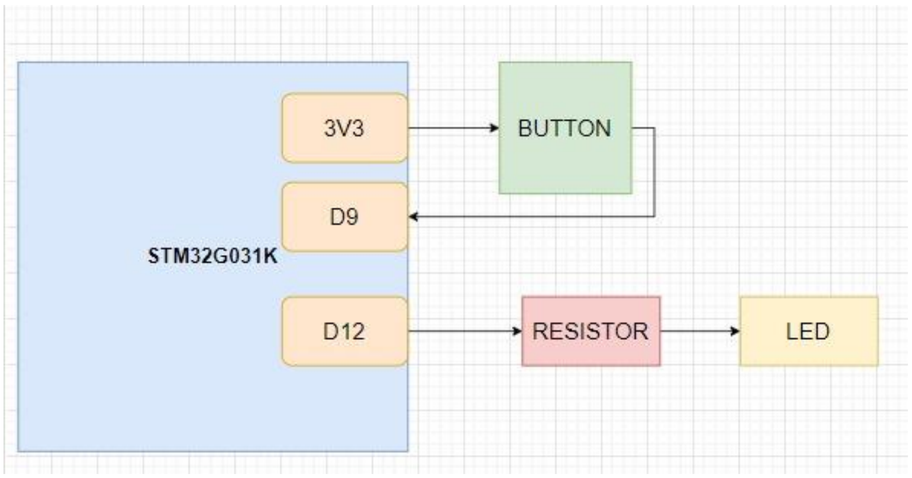


Figure 11: Problem 4 Block Diagram

```
// Q4.s
 // Author: 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 IDR 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, 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 PB4 and PB5 for button and led 01 and 00 in MODER
     ldr r6, =GPIOB_MODER
     ldr r5, [r6]
     // cannot do with movs, so use pc relative
     ldr r4, =[0x30300]
     bics r5, r5, r4
     ldr r4, =[0xCFDFF]
     orrs r5, r5, r4
     str r5, [r6]

     // turn on led connected PB5 if button is pressed
loop:
     ldr r6, = GPIOB_IDR
     ldr r5, [r6] //IDR Value
     lsrs r5, r5, #8 // Shifting to first bit
     movs r4, #0x1
```

```
ands r5, r5, r4 //Getting the value of button pressed or not

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

//If is equal
BEQ:
ldr r6, = GPIOB_ODR
ldr r5, [r6] //ODR Value
movs r4, 0x10
orrs r5, r5, r4 //Setting led on
str r5, [r6]
b loop

//If is not equal
BNE:
ldr r6, = GPIOB_ODR
ldr r5, [r6] //ODR Value
movs r4, 0x10
bics r5, r5, r4 //Setting led off
str r5, [r6]
b loop

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

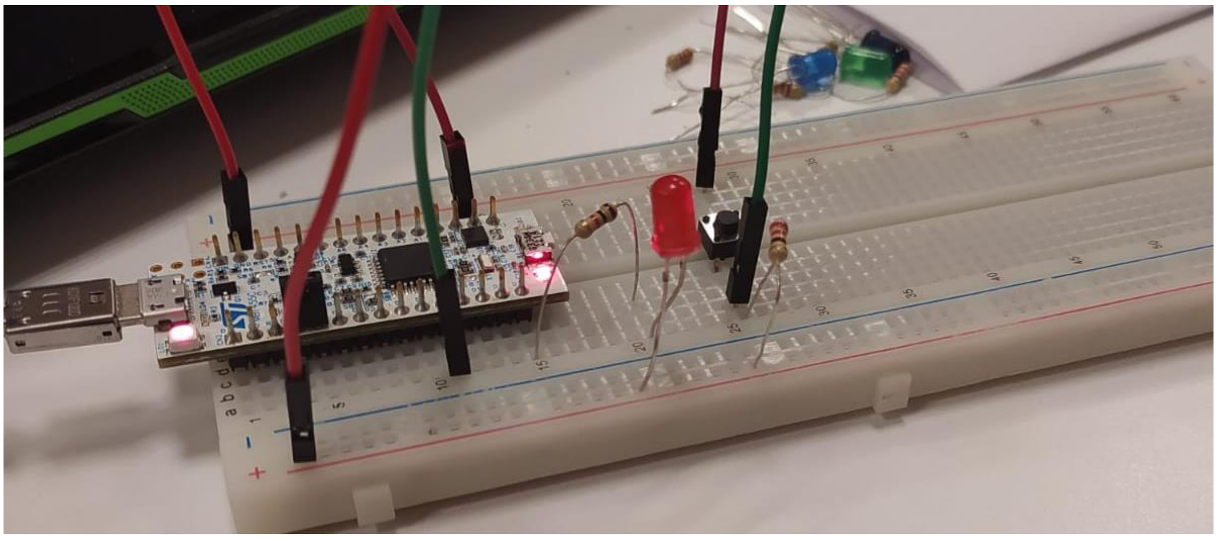


Figure 12: When push button

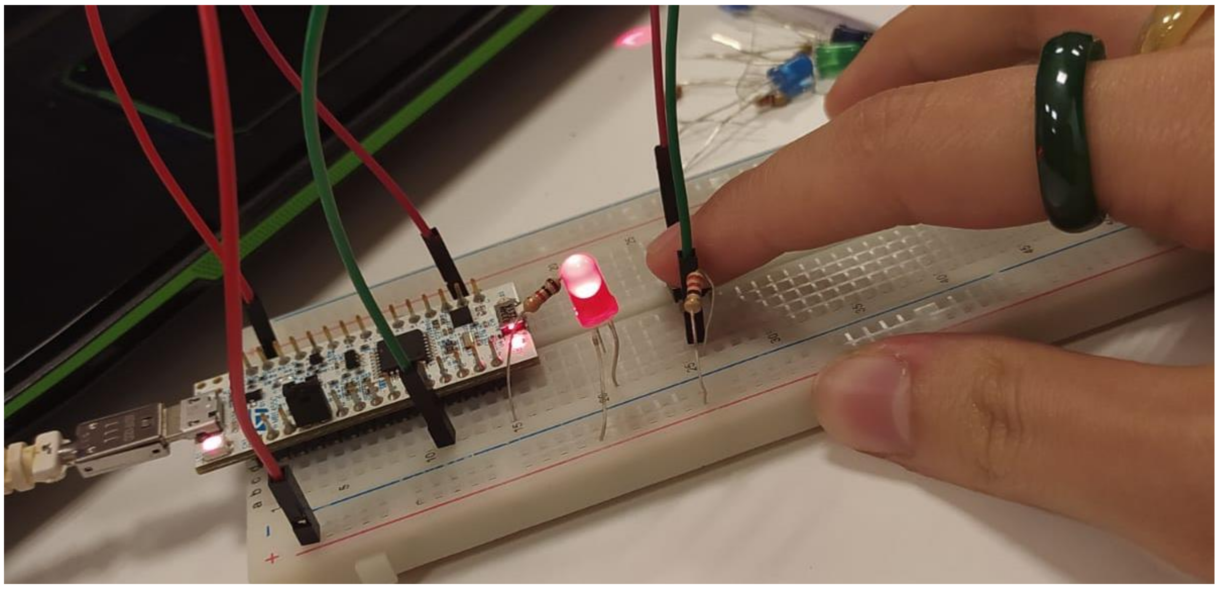


Figure 13: When not push button

- Using an oscilloscope, capture your button press, and see if you can spot any bouncing happening. Include a picture of this setup and capture in your report and comment on the result. Try with different buttons if you have more than one and compare the results.

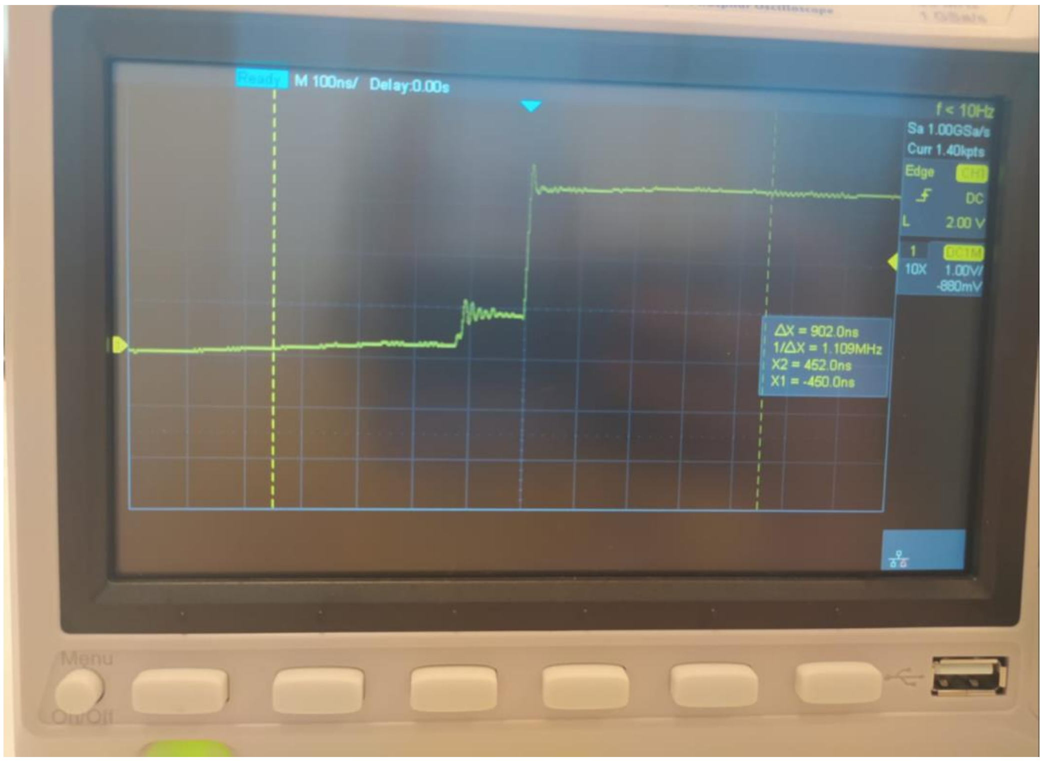


Figure 14: Voltage signal on the led button 1 circuit

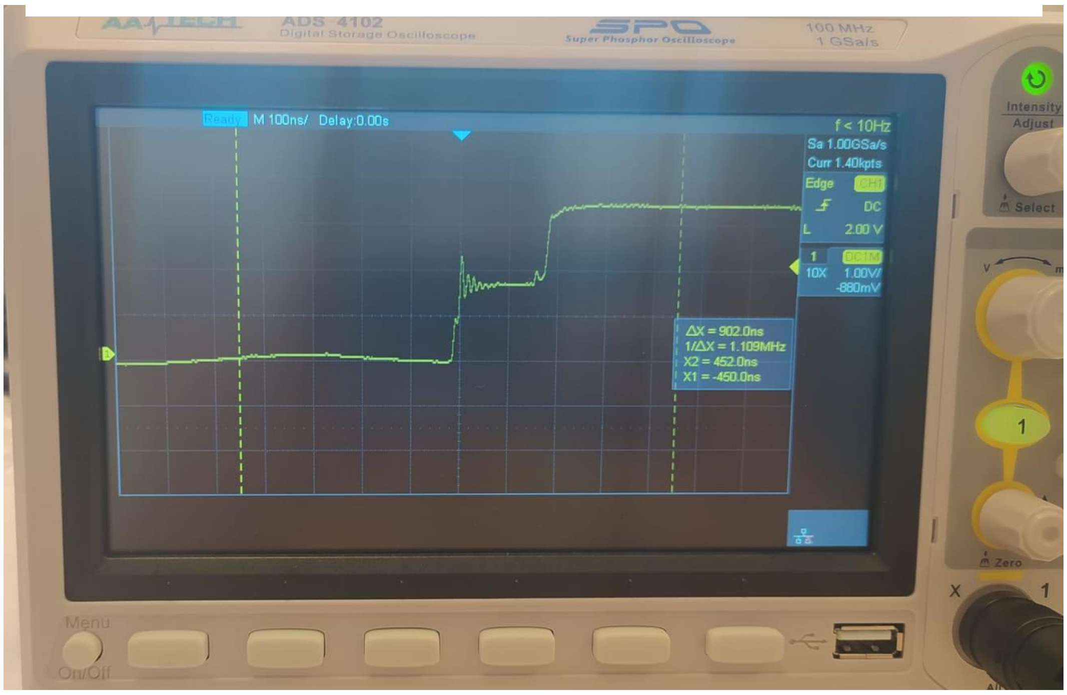


Figure 15: Voltage signal on the led button 2 circuit

When the button is not pressed, the value dec the anode and cathode of the led is 0 volts. When the button is pressed, this value will change from 0 volts to 3.3 volts. But since the state of the button is not stable during this transition, it dec to a random volt value for a short time between 0 and 3.3 volts

Problem 5

In this problem you are asked to write code that will blink 1 external LED at roughly 1 second intervals.

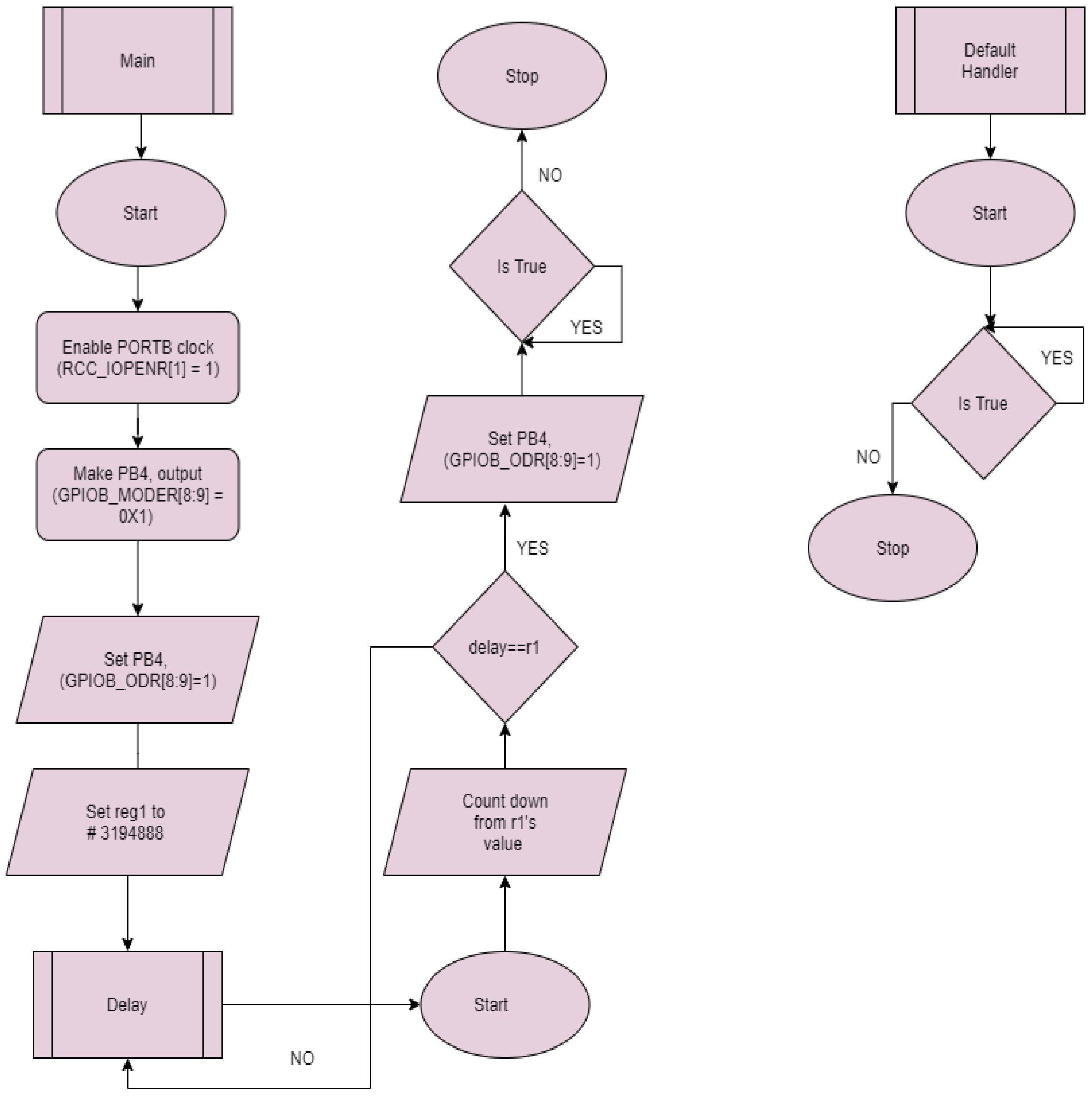

Main
Start
Enable PORTB clock (RCC_IOPENR[1] = 1)
Make PB4, output (GPIOB_MODER[8:9] = 0X1)
Set PB4, (GPIOB_ODR[8:9]=1)
Set reg1 to # 3194888
Delay
NO
Start
Count down from r1's value
delay==r1
YES
Set PB4, (GPIOB_ODR[8:9]=1)
Is True
YES
NO
Stop
Default Handler
Start
Is True
YES
NO
Stop

Figure 16: Problem 5 Flowchart

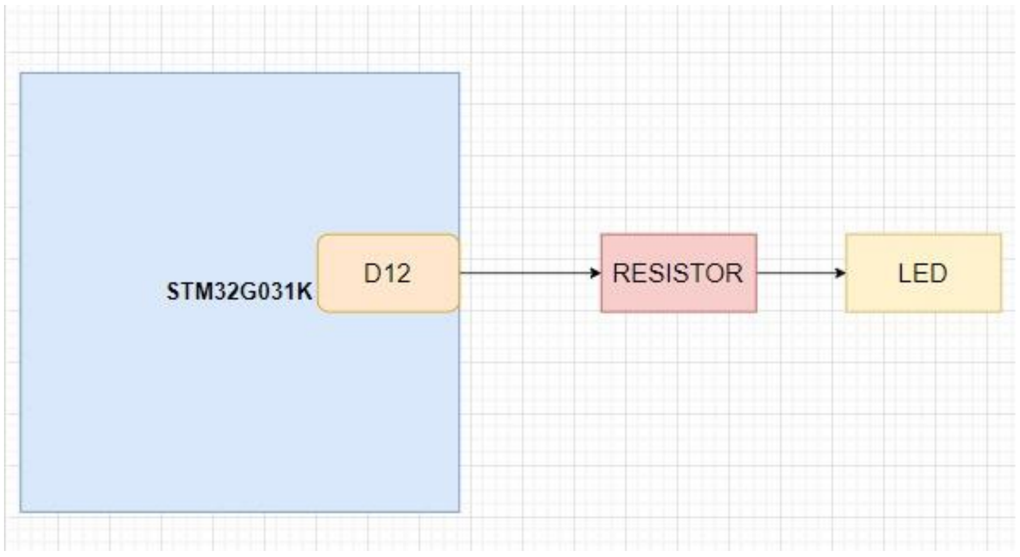


Figure 17: Problem 5 Block Diagram

```
/*

 * asm.s

 *

 * author: Furkan Cayci

 *

 * 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


/* 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 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 PC6 for led 01 for bits 12-13 in MODER */
        ldr r6, =GPIOB_MODER
        ldr r5, [r6]
        /* cannot do with movs, so use pc relative */
        movs r4, 0x3
        lsls r4, r4, #8
        bics r5, r5, r4
        movs r4, 0x1
        lsls r4, r4, #8
        orrs r5, r5, r4
        str r5, [r6]

        /* turn on led connected to C6 in ODR */
        ldr r6, =GPIOB_ODR
        ldr r5, [r6]
        movs r4, 0x1

        lsls r4, r4, #4

        orrs r5, r5, r4

        str r5, [r6]


loop:

        //turn on led connected to PB4 in ODR

        ldr r6, =GPIOB_ODR

        ldr r5, [r6]

        movs r4, 0x1

        lsls r4, r4, #4

        orrs r5, r5, r4

        str r5, [r6]


        ldr r1, =#1000000

        bl delay

        //turn off led connected to PB4 in ODR

        ldr r6, =GPIOB_ODR

        ldr r5, [r6]

        movs r4, 0x1

        lsls r4, r4, #4

        bics r5, r5, r4

        str r5, [r6]


        ldr r1, =#10000000

        bl delay
        b loop


delay:

        subs r1, r1, #1

        bne delay

        bx lr


        //this should never get executed

        nop
```

- Using an oscilloscope, capture and measure the blink interval. This should be around 1 sec.

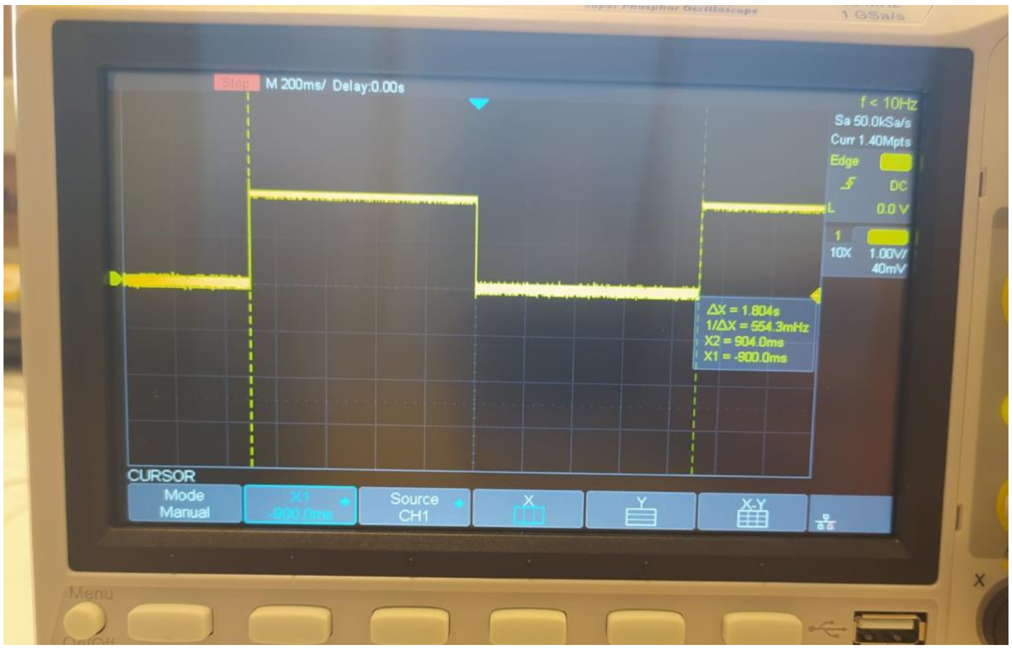


Figure 18: Voltage signal on the led

It has been observed that the image appears when measuring the voltage signal on the external led leg in the blink circuit is about 2 volts when the led is on and 0 volts when it goes out. The time/div ratio was set to 200ms, and it was observed that the led burned out 980ms and went out 980ms. Then the code was changed and the led was set to light 1 time per second.

- Try to estimate the CPI (Cycles per Instruction) and comment/justify the results.

$$CPI = \frac{5 * 43 + 4 * 12 + 4 * 31 + 3 * 12}{100}$$

$$= 4.23$$

Conclusion

In this project, the functions of the components on the stm32g031k card were written first. Then, the ld3 led on the board was burned by giving high to the PC6 pin. The high and low states of the PC6 pin were measured with a multimeter. Then, 4 external LEDs were connected to different pins and lit. It has been observed here that the codes are not cleared when the power is turned on again after the power is turned off. Then an external LED was lit thanks to a button. It has been observed that when the button is pressed, the led lights up, and when it is not pressed, the led does not light up. Here, the voltage signal between the anode and the cathode of the led was observed by pressing and pulling the button with the help of an oscilloscope. When the voltage signal is examined, since the button changes from 0 to 1, the signal deviates to random voltages at intermediate values. Finally, the voltage signal between the anode and cathode of the led was measured by flashing an external led once per second. Here the voltage is 0 volts when the led is off. When the LED is lit, the voltage value of the LED is seen as 2 volts. The duration of these values is 980 ms. ie 980ms in high mode and 980ms in low mode. Then, by playing on the code, these times decreased and the led started to blink once a second.

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