GEBZE TECHNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC335 - MICROPROCESSORS LABORATORY

LAB #1

Muhammet Cemal Eryiğit	1801022024
Şahabettin Alpcan Soydaş	1801022014
Mert Tuncay Firil	1901022285

1) In this section, STM32 Nucleo G031K8 board will be identified. All the components, peripherals, and pin connections will be explained about their usage.

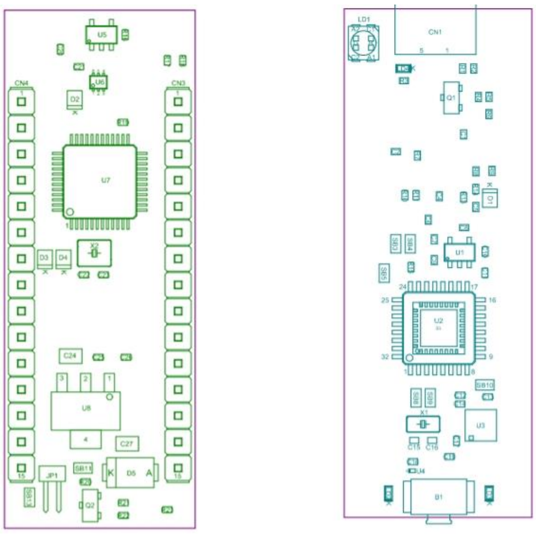

Figure 1: STM32 G031K8 board top and bottom layouts

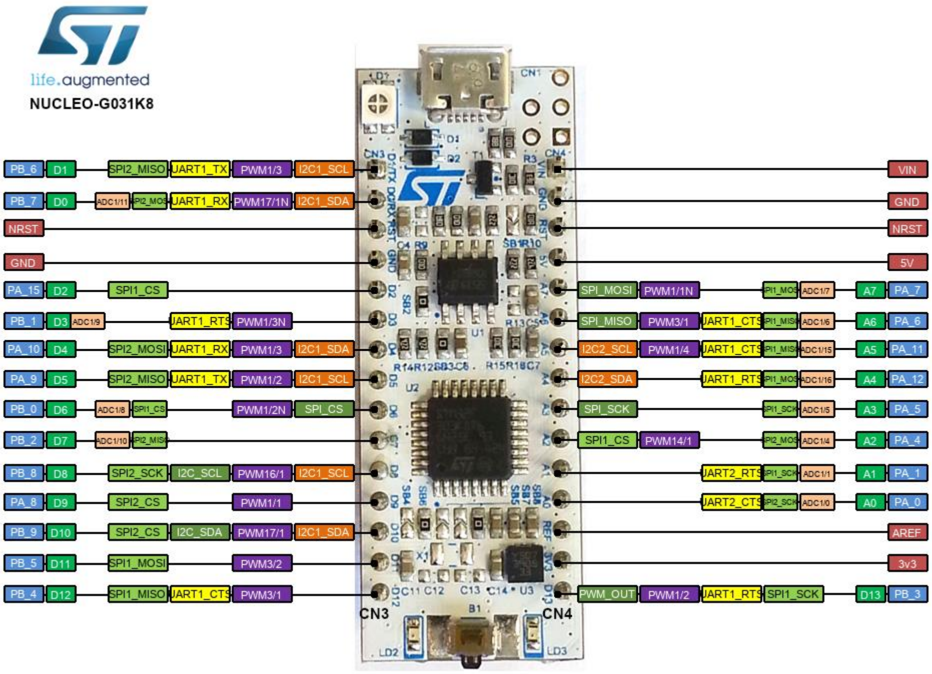

Figure 2: STM32 G031K8 Input Output Pins

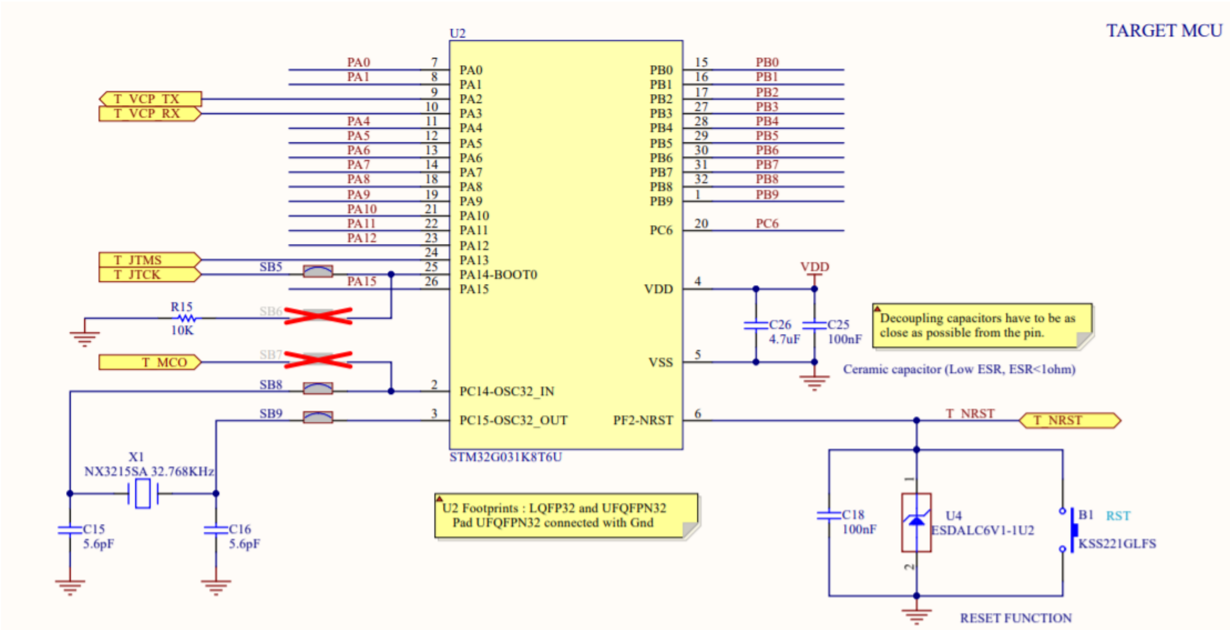

Figure 3: Target MCU Schematic

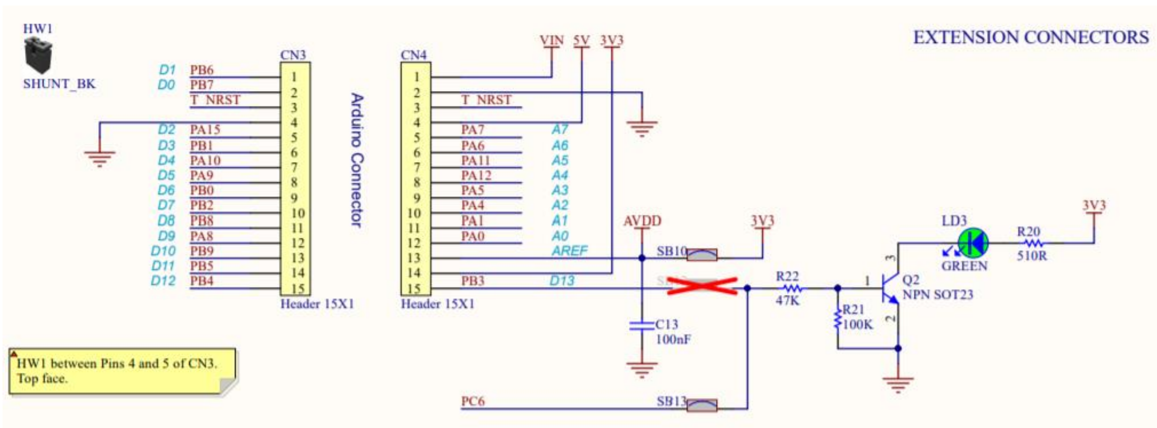

Figure 4: Extension Connectors

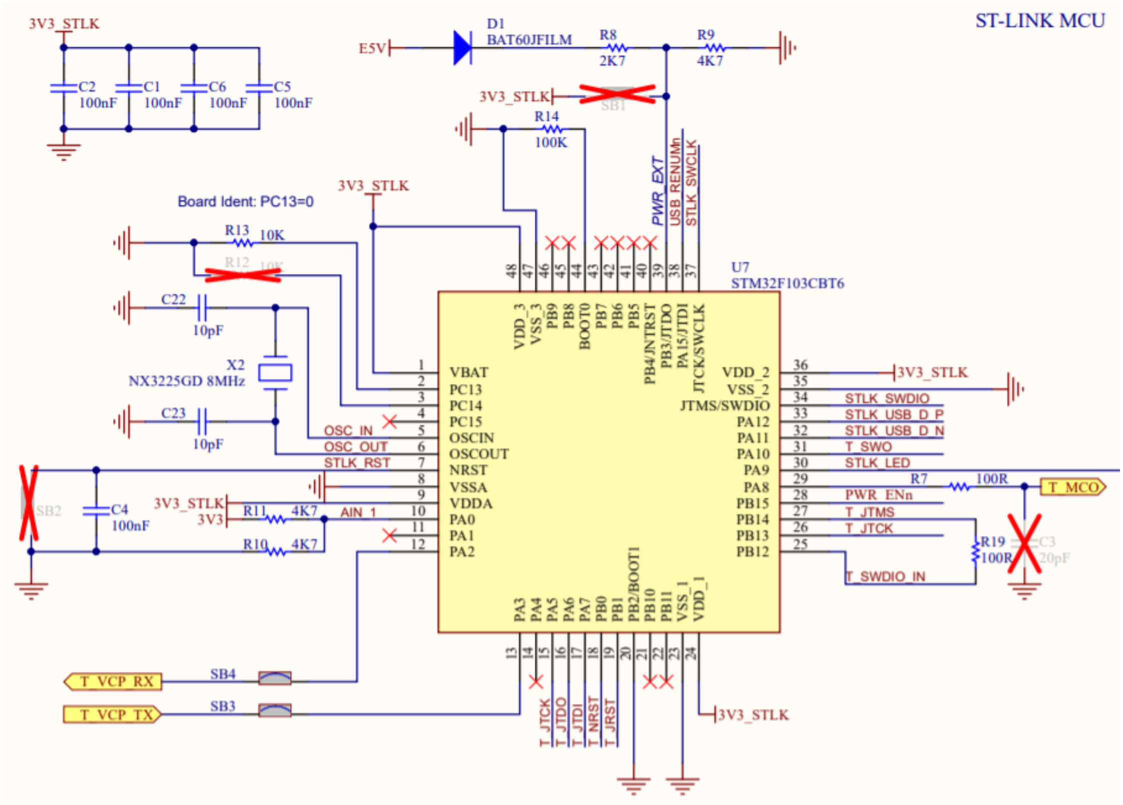

Figure 5: Second Microcontroller for ST-LINK Connection

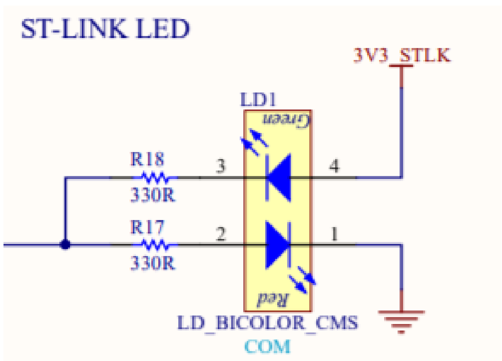

Figure 6: ST-LINK LED that connected to PA9

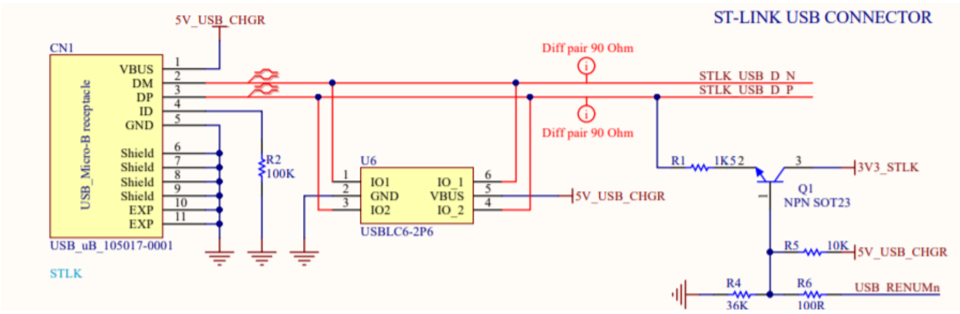

Figure 7: ST-LINK USB Connector

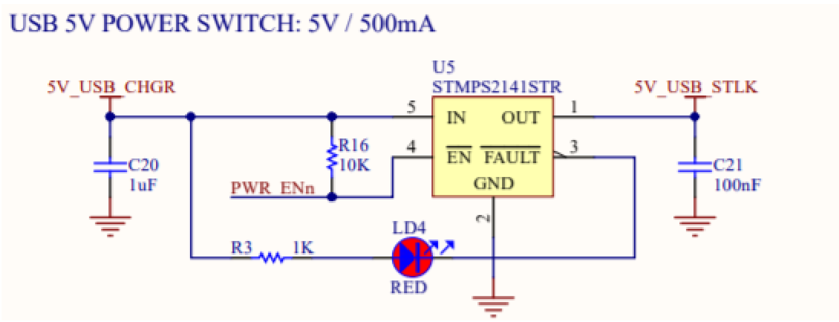

Figure 8: USB 5V Power Switch

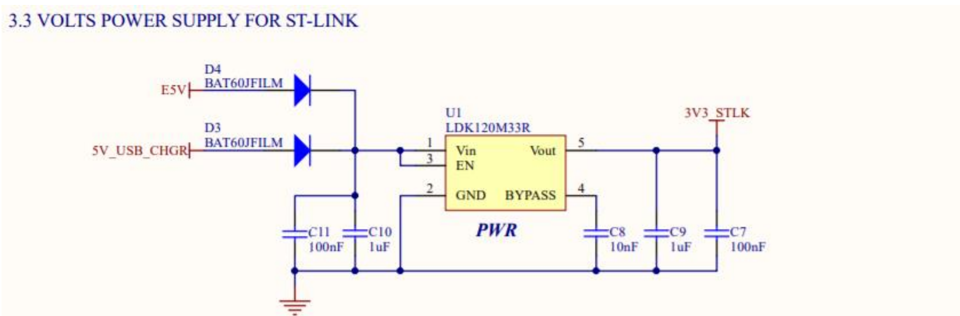

Figure 9: 3.3 Volts Power Supply for ST-LINK

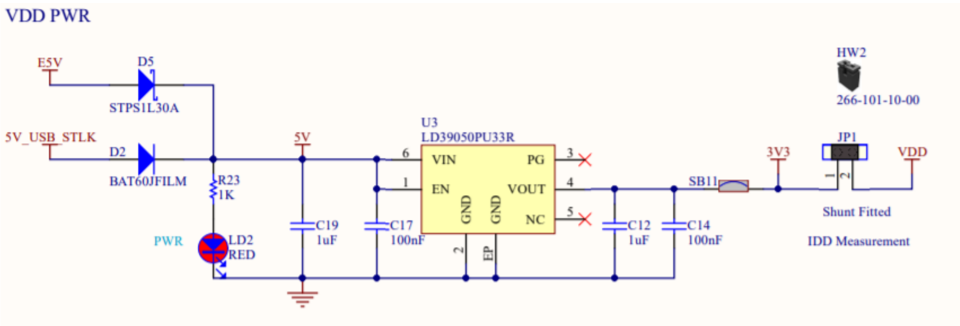

Figure 10: Vdd Power

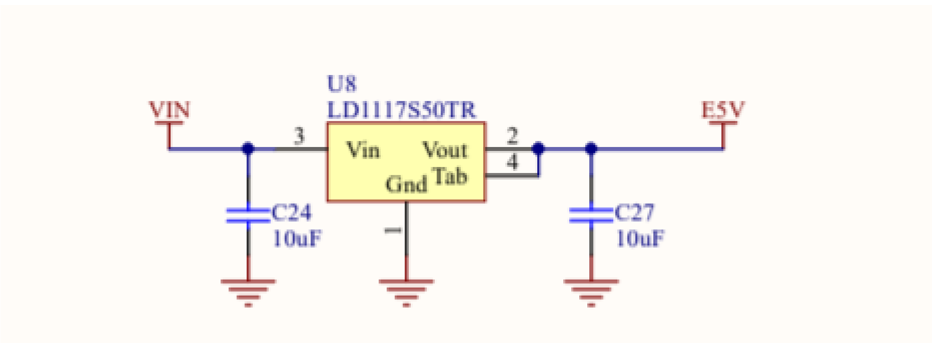

Figure 11: 5V Vin Power

For STM32 G031K8, all on-board schematics are can be seen in the above. Also figure 2 shows us the all input output pins.

2) In this problem, you are asked to code that will light up the on-board LED connected to pin PC6. Multimeter will be used to measure PC6 pin when the LED is on, and when the LED is off. Nominal voltage and maximum voltage values of our microcontroller pins will be defined.

There is no connection diagram for problem 2. Because in problem 2, the on-board PC6 LED is used.

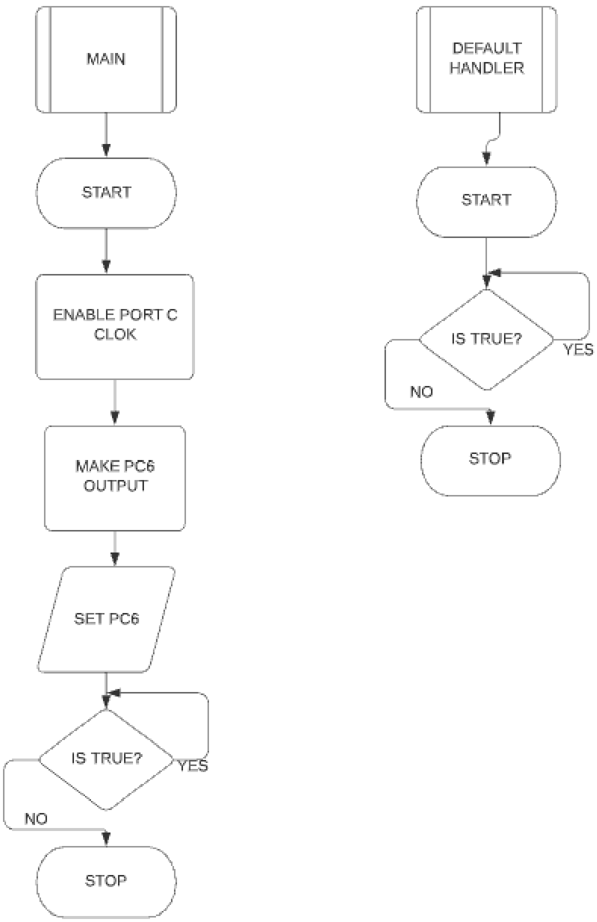

Figure 12: Flowchart of the problem 2

```
.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

.equ RCC_BASE,         (0x40021000)          // RCC base
address=0x40021000 RMS0444 page57 Bus=AHB
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR
address=0x34 page174 IO Port clock enable register RCC_IOPENR

.equ GPIOC_BASE,       (0x50000800)          // GPIOC base
address=0x50000800 RMS0444 page57 Bus=IOPORT
.equ GPIOC_MODER,      (GPIOC_BASE + (0x00)) // GPIOC MODER
address=0x00 page205 GPIOx_MODER
.equ GPIOC_ODR,        (GPIOC_BASE + (0x14)) // GPIOC ODR
address=0x14 page207 GPIOx_ODR

/* 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] //r5 HOLDS RCC_IOPENR REGISTER DATA
/* movs expects imm8, so this should be fine */
//RCC_IOPENR's second bit must be "1" to enable GPIOC clock
movs r4, 0x4
```

```
orrs r5, r5, r4 // orr 0100 to make second bit of RCC_IOPENR
str r5, [r6] //store r5 reg data to address r6

/* setup PC6 for led 01 for bits 12-13 in MODER */
ldr r6, =GPIOC_MODER //r6 holds the GPIOC_MODER ADDRESS
ldr r5, [r6] //r5 holds GPIOC_MODER DATA
/* cannot do with movs, so use pc relative */
movs r4, 0x3 //rd holds 0011
lsls r4, r4, #12  //logical shift left 12 =>0011_0000_0000_0000
bics r5, r5, r4 //bitclear
movs r4, 0x1 //0001
lsls r4, r4, #12 // bit[13:12]"01"
orrs r5, r5, r4 // orr with r5
str r5, [r6] // store r5 data to GPIOC_MODER


/* turn on led connected to C6 in ODR */
ldr r6, =GPIOC_ODR // r6 holds GPIOC_ODR address
ldr r5, [r6] //r5 holds GPIOC_ODR data
movs r4, 0x40 //0100_0000 GPIOC_ODR 6. bit must be 1 for PC6
orrs r5, r5, r4 // orr with 0100_0000
str r5, [r6] //store r5 data to GPIOC_ODR address

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

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

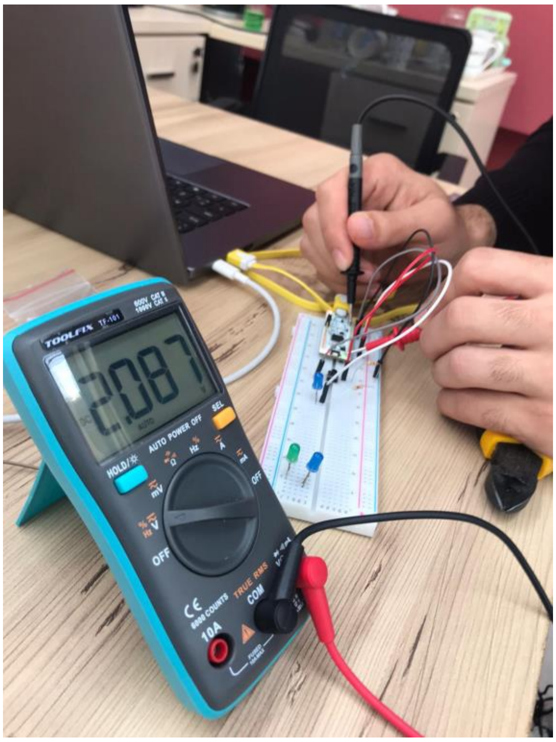

Figure 13: Voltage between PC6 LED’s pin when it is on

Firstly, flowchart of the problem 2(Figure X) is created. STM32 G031K8 reference manual is read carefully to learn which addresses and registers we need to light up the LED. PC6 on-board LED is chosen to light up. GPIOC port address declarations are defined. GPIOC_ODR register is set to light up the PC6 LED. Then voltage of the PC6 LED is measured using multimeter as can be seen in the Figure 13.

3) In this problem you are asked to write code that will light up 4 external LEDs connected to the board. Power off your board, then power back on. Are the LEDs light up again? Explain and justify the results.

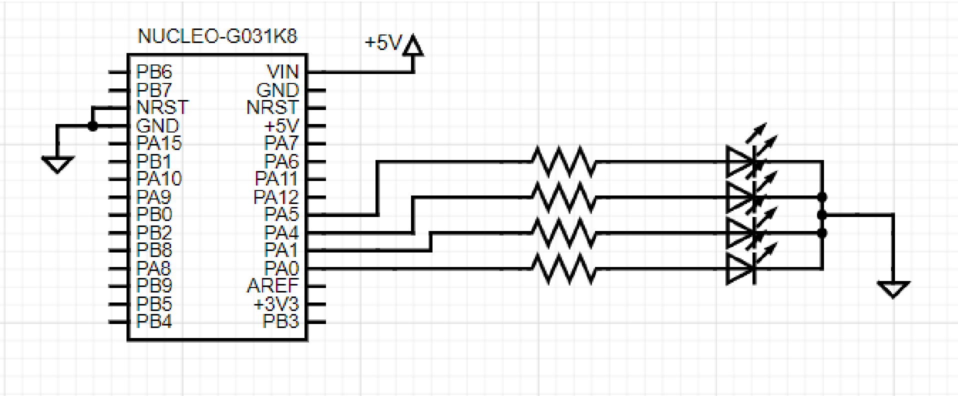

Figure 14: Problem 3 Connection Diagram

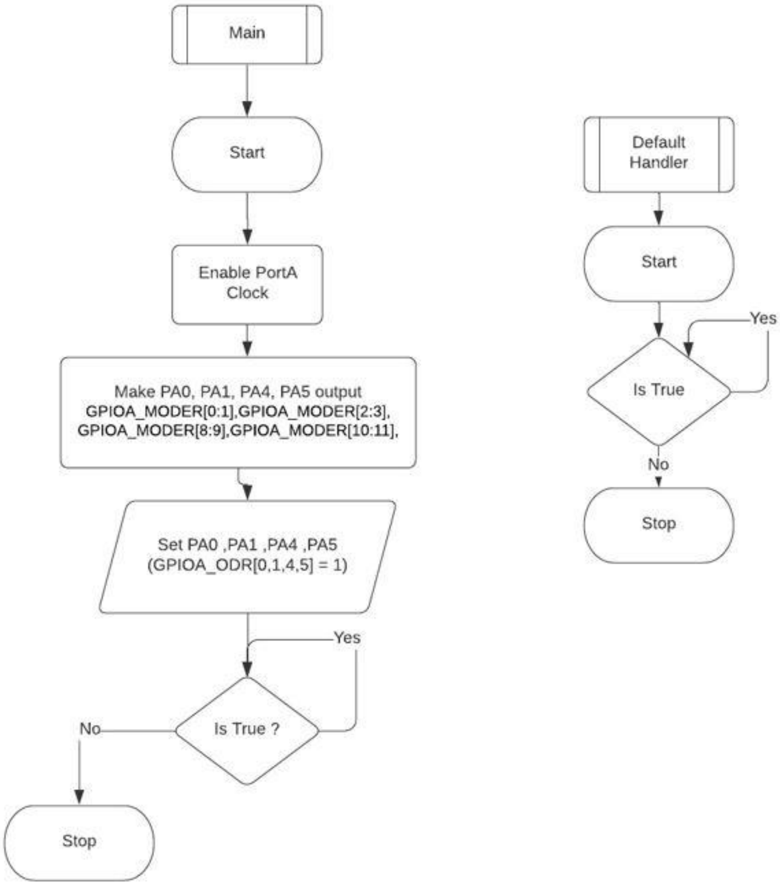

Figure 15: Flowchart of the problem 3

```
.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 GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER
register offset
.equ GPIOA_ODR,        (GPIOA_BASE + (0x14)) // GPIOA 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, 0x1
orrs r5, r5, r4
str r5, [r6]

ldr r6, =GPIOA_MODER //r6 holds the GPIOA_MODER ADDRESS
ldr r5, [r6] //r5 holds GPIOA_MODER DATA
/* cannot do with movs, so use pc relative */
movs r4, 0xF // r4 holds 1111 for GPIOA_MODER[0:1],
GPIOA_MODER[2:3](reset mode)
movs r3, 0xF // r3 holds 1111 for GPIOA_MODER[8:9],
GPIOA_MODER[10:11]
lsls r3, #8 //logical shift left 8
bics r5, r5, r4 //bitclear r4
bics r5, r5, r3 //bitclear r3
movs r4, 0x5 r4 holds 0101 (gpio mode)
movs r3, 0x5 r3 holds 0101 (gpio mode)
lsls r3, #8 r3 holds 1111 for GPIOA_MODER[8:9]
orrs r5, r5, r3
orrs r5, r5, r4
str r5, [r6] // store r5 data to GPIOA_MODER

/* turn on led connected to A0,A1,A4,A5 in ODR */
ldr r6, =GPIOA_ODR // r6 holds GPIOA_ODR address
ldr r5, [r6] //r5 holds GPIOA_ODR data
movs r4, 0x33 //r4 holds 0011 0011 for A0,A1,A4,A5
orrs r5, r5, r4
str r5, [r6] //r5 holds GPIOA_ODR data

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

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

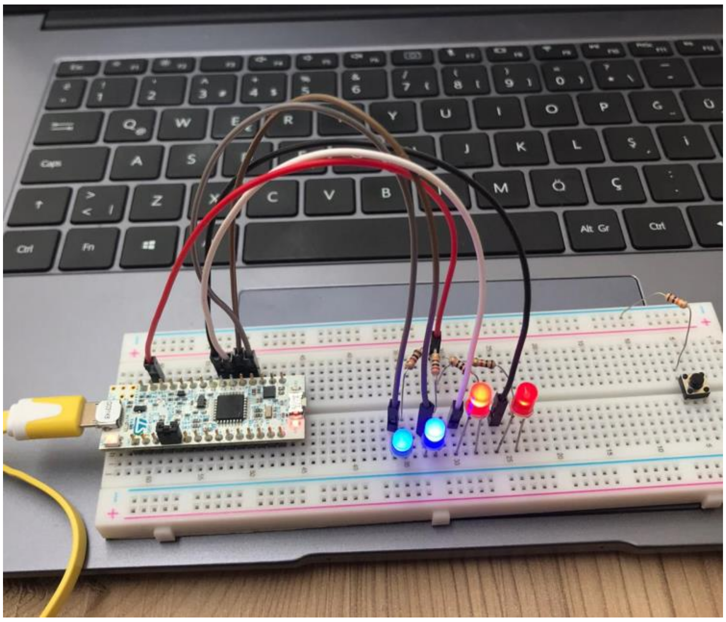

Figure 16: 4 external led light

We connected 4 LEDs to the GPIO ports. When digital output is given to the ports, the leds started to light. (Figure 3.1) The written embedded code is kept in the flash part of the microcontroller. When we turn the microcontroller off and on again, the same ports continue to work. Leds are on again

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. 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.

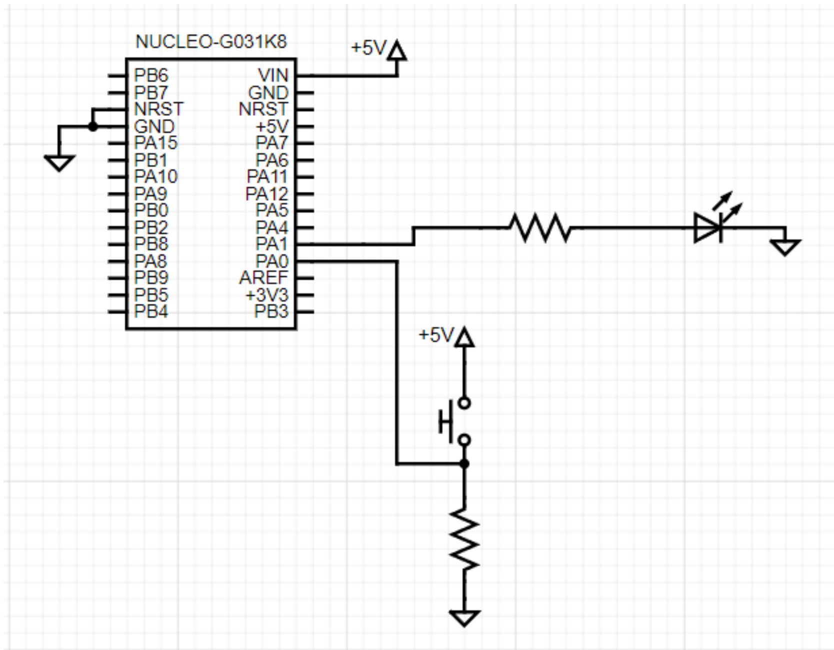

Figure 17: Problem 4 Connection Diagram

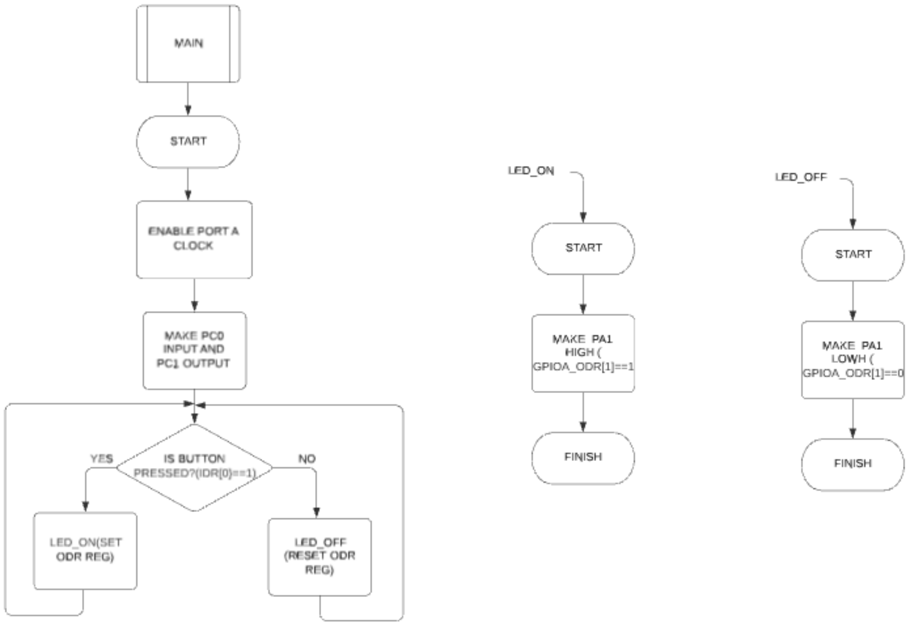

Figure 18: Flowchart of the problem 4

```
.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 GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER
register offset
.equ GPIOA_ODR,        (GPIOA_BASE + (0x14)) // GPIOA ODR register
offset
```

```
.equ GPIOA_IDR,        (GPIOA_BASE +(0x10)) // GPIOA_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 GPIOA clock, bit2 on IOPENR */
ldr r6, =RCC_IOPENR
ldr r5, [r6]
/* movs expects imm8, so this should be fine */
movs r4, 0x1 //A PORT ACTIVE
orrs r5, r5, r4
str r5, [r6]

/* setup PA0 and PA1 for led and button */
ldr r6, =GPIOA_MODER
ldr r5, [r6]

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

movs r4, 0xF
bics r5, r5, r4
movs r4, 0x4 //PA0-INPUT PA1-OUTPUT => 0100 =>0x4
orrs r5, r5, r4
str r5, [r6] // store r5 data to GPIOC_MODER //PA0-Button PA1-LED
```

```
button: //connected to PA0
ldr r6, =GPIOA_IDR
ldr r5, [r6]
lsrs r5, r5, #0
movs r4, #0x1 //r4=0x1
ands r5, r5, r4 // GPIOA_IDR and r4
cmp r5, #0x1
bne led_off // if not branch to led_off
beq led_on // if GPIOA_IDR[0]==1
led_on:
ldr r6, =GPIOA_ODR
ldr r5, [r6]
movs r4, #0x2 //led connected to PA1 0010=>0x2
orrs r5, r5, r4
str r5, [r6]
b button
led_off:
ldr r6, =GPIOA_ODR
ldr r5, [r6]
movs r4, #0
ands r5, r5, r4 //reset the ODR reg
str r5, [r6]
b button
/* for(;;); */
//b . // always branch to button label

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

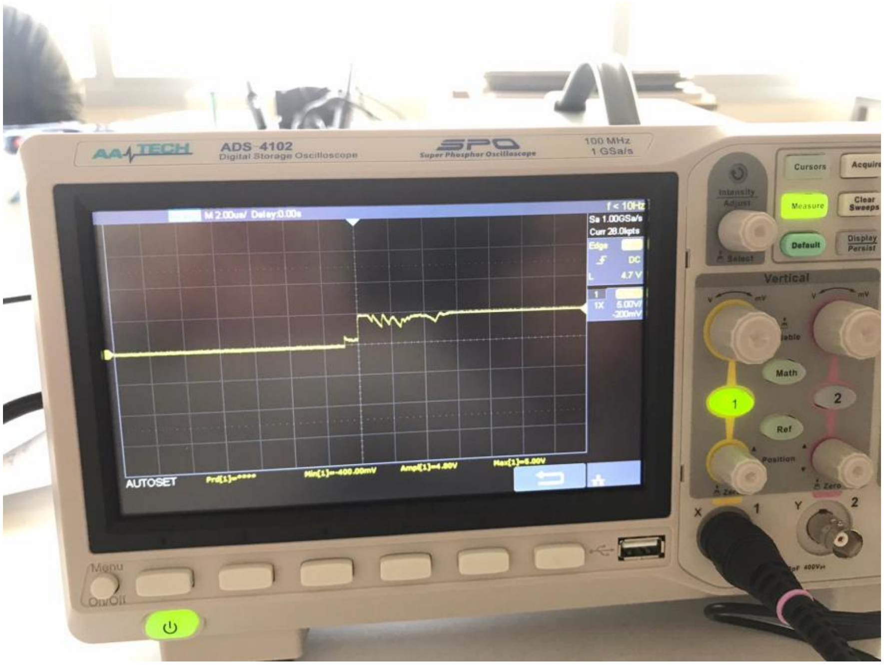

Figure 19: Observing the bouncing on the button

Firstly, flowchart of the problem 4(Figure X) is created. GPIOA port is chosen to connect button as an input and LED as an output. Button is connected to PA0 and LED is connected to PA1. GPIOA clock and GPIOA register address declarations are defined. GPIOA_MODER register is used to define input/output components. GPIOA_IDR register holds the button value. PA1 LED’s on/off status is chosen by GPIOA_IDR register. LED_OFF and LED_ON functions are defined. When button is pressed or not pressed code branchs to LED_OFF or LED_ON functions. Also button bouncing is observed in the laboratory.

5) In this problem you are asked to write code that will blink 1 external LED at roughly 1 second intervals.Using an oscilloscope, capture and measure the blink interval. This should be around 1 sec.Try to estimate the CPI (Cycles per Instruction) and comment/justify the results.

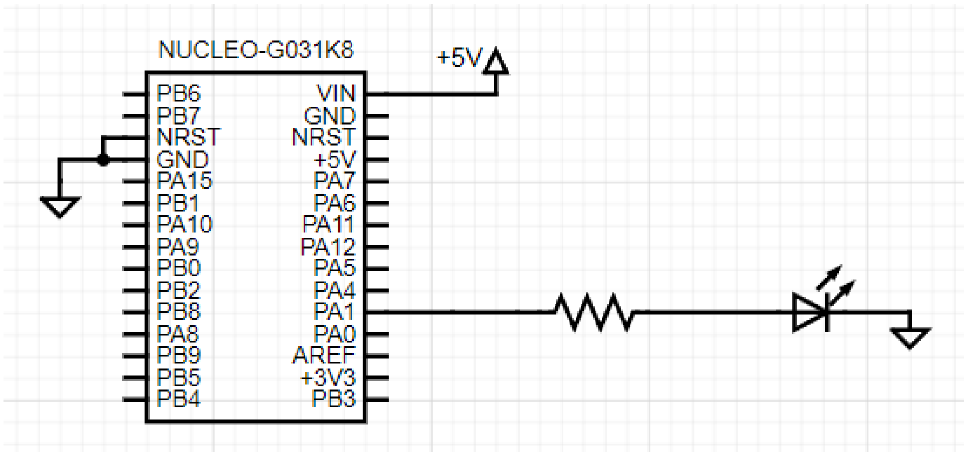

Figure 20: Problem 5 Connection Diagram

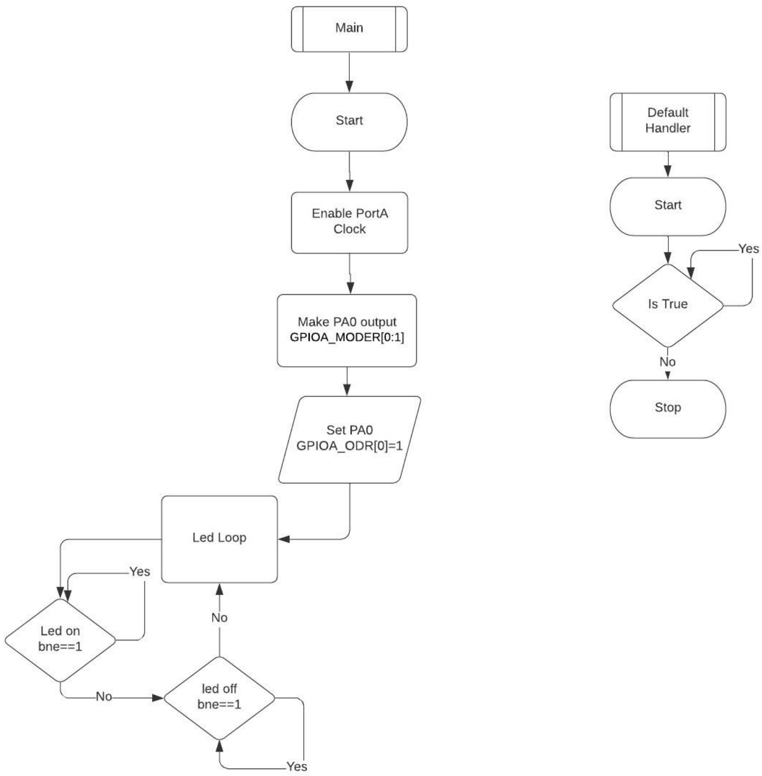

Figure 21: Flowchart of the problem 5

```
.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


.equ RCC_BASE,         (0x40021000)          // RCC base address
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR
address=0x34 page174 IO Port clock enable register RCC_IOPENR

.equ GPIOA_BASE,       (0x50000000)          // GPIOC base address
.equ GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOC MODER
register offset
.equ GPIOA_ODR,        (GPIOA_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 GPIOA clock, bit2 on IOPENR */
ldr r6, =RCC_IOPENR
ldr r5, [r6] //r5 HOLDS RCC_IOPENR REGISTER DATA
/* movs expects imm8, so this should be fine */
movs r4, 0x1 // r4 holds  0001
orrs r5, r5, r4 // orr 0100 to make second bit of RCC_IOPENR
str r5, [r6] //store r5 reg data to address r6

/* setup PC6 for led 01 for bits 12-13 in MODER */
ldr r6, =GPIOA_MODER //r6 holds the GPIOA_MODER ADDRESS
ldr r5, [r6] //r5 holds GPIOA_MODER DATA
/* cannot do with movs, so use pc relative */
movs r4, 0x3 // r4 holds 0011
bics r5, r5, r4 //bitclear
movs r4, 0x1 // r4 holds 0001
orrs r5, r5, r4 // orr with r5
str r5, [r6] // store r5 data to GPIOA_MODER

ldr r6, =GPIOA_ODR // r6 holds GPIOA_ODR address
ldr r5, [r6] //r5 holds GPIOA_ODR data
led_loop: //branch
     movs r4, 0x1 // r4 holds 0001
orrs r5, r5, r4 // orr with r5
str r5, [r6] //r5 holds GPIOA_ODR data

ldr r2,=0x28B0AA //r2 holds decimal 2666666 (for 1 sn blinking)
led_on:  //branch
subs r2,r2, #1 //decrease 1 from r2
bne led_on //go to led_on branch (if branch not equal)

movs r4, 0x0 //r4 holds 0000
ands r5, r5, r4 // and with r5
str r5, [r6] //r5 holds GPIOA_ODR data (data reset)

ldr r2,=0x28B0AA //r2 holds decimal 2666666 (for 1 sn blinking)
led_off:  //branch
subs r2,r2, #1 //decrease 1 from r2
bne led_off //go to led_off branch (if branch not equal)

b led_loop //go to led_loop branch

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

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

We used an oscilloscope to observe the period of the external led flashing code written. The oscillator of the microprocessor has a standard operating speed of 8 MHz. But when it is set to 8 Mhz, 1 period takes 1.5 s. This is because the oscillator is restricted to 2/3 of its operating frequency. (Default (figure 5.2 and figure 5.3)). We set the operating frequency to be 5.33 MHz, with 1 period of 1 sec. (Figure 5.4)

Table 39. HSE oscillator characteristics(1)

Symbol	Parameter	Conditions(2)	Min	Typ	Max	Unit
f_{OSC_IN}	Oscillator frequency	-	4	8	48	MHz

Figure 22: stm32g0_datasheet_page_61

1. Guaranteed by design.
2. Resonator characteristics given by the crystal/ceramic resonator manufacturer.
3. This consumption level occurs during the first 2/3 of the $t_{SU(HSE)}$ startup time
4. $t_{SU(HSE)}$ is the startup time measured from the moment it is enabled (by software) to a stabilized 8 MHz oscillation is reached. This value is measured for a standard crystal resonator and it can vary significantly with the crystal manufacturer

Figure 23: Stm32g0_datasheet_page_62

Figure 24: Oscilloscope image

1 second is given for the LED to turn on and off. Since SUBS and BNE commands take 1 cycle, we need to repeat 2666666x2 times for the crystal running at 2/3 of it 5.33MHz. The hexadecimal equivalent is 28B0AA.

CONCLUSION

In this laboratory, STM32 G031K8 MCU board is programmed using Assembly. LDR, MOVS, ADDS, LSLS etc. Assembly commands are used to write data to registers. RCC and GPIO addresses are defined for each problem. GPIO_MODER, GPIO_ODR and GPIO_IDR registers are used for input/output processes. Reading information from reference manual and implemantation to the microcontroller is learnt. After coding, some measurements from board and the connected components are taken using oscilloscope and multimeter.

REFERENCES:

1) https://github.com/fcayci/stm32g0

2) https://www.st.com/resource/en/datasheet/stm32g031k8.pdf

3) https://www.st.com/resource/en/errata_sheet/dm00625293-stm32g031x4x6x8-device-errata-stmicroelectronics.pdf

4) 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

5) https://www.st.com/resource/en/reference_manual/dm00371828-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf