GEBZE TECHNICAL UNIVERSITY

ELEC335- MICROPROCESSORS LAB
FALL 2021

LAB 1

PROBLEM 1

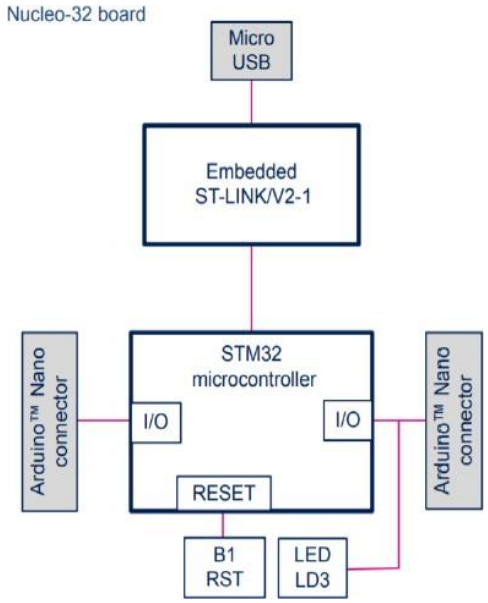

Figure 1. Hardware Block Diagram

STM32G031K8Tx Features

- Core: Arm® 32-bit Cortex®-M0+ CPU, frequency up to 64 MHz
- Up to 64 Kbytes of Flash memory with protection and securable area
- 8 Kbytes of SRAM with HW parity check
- Voltage range: 1.7 V to 3.6 V
- Power-on/Power-down reset (POR/PDR)
- Low-power modes: Sleep, Stop, Standby, Shutdown
- 4 to 48 MHz crystal oscillator and 32 kHz crystal oscillator with calibration
- -40°C to 85°C/105°C/125°C operating temperature

STM32G031K8Tx Functional Overview

- Cortex®-M0+ core with MPU
- Memory protection unit
- Embedded Flash memory
- Embedded SRAM
- Power supply management
- Clocks and startup
- Cyclic redundancy check calculation unit (CRC)

- General-purpose inputs/outputs (GPIOs)
- Direct memory access controller (DMA)
- DMA request multiplexer (DMAMUX)
- Interrupts and events
- Analog-to-digital converter (ADC)
- Voltage reference buffer (VREFBUF)
- Real-time clock (RTC), tamper (TAMP) and backup registers
- Serial peripheral interface (SPI)
- Universal synchronous/asynchronous receiver transmitter (USART)
- Low-power universal asynchronous receiver transmitter (LPUART)
- Timers and watchdogs
- 11 timers (one 128 MHz capable): 16-bit for advanced motor control, one 32-bit and four 16-bit general-purpose, two low-power 16-bit, two watchdogs, SysTick timer
- Calendar RTC with alarm and periodic wakeup from Stop/Standby/Shutdown
- Communication interfaces

  – Two I2C-bus interfaces supporting Fastmode Plus (1 Mbit/s) with extra current sink, one supporting SMBus/PMBus and wakeup from Stop mode

  – Two USARTs with master/slave synchronous SPI; one supporting ISO7816 interface, LIN, IrDA capability, auto baud rate detection and wakeup feature

  – One low-power UART

  – Two SPIs (32 Mbit/s) with 4- to 16-bit programmable bit frame, one multiplexed with I2S interface
- Development support: serial wire debug (SWD)

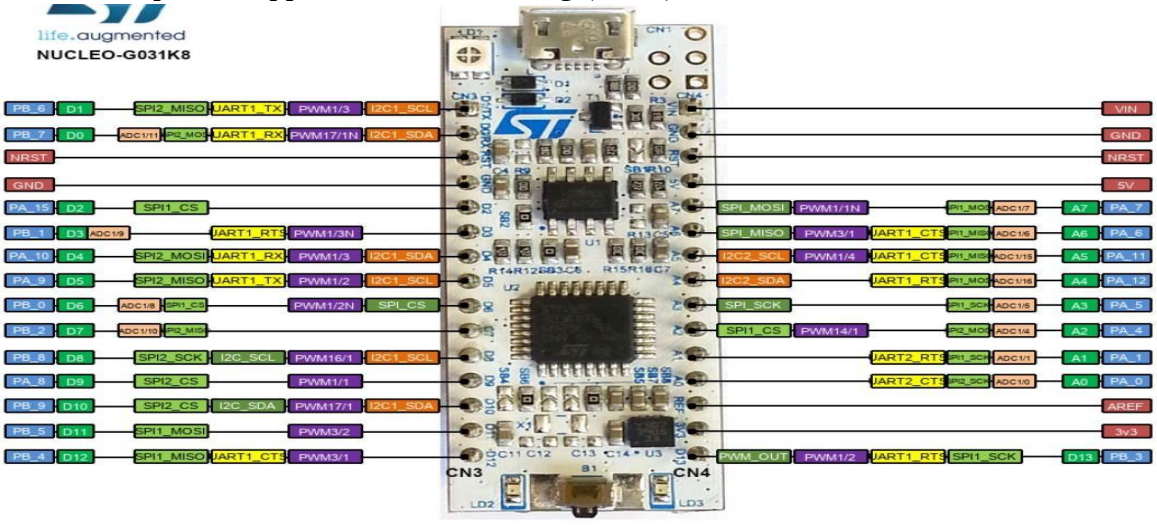

Figure 2. Board Pinout

ST-LINK/V2-1

- The ST-LINK/V2-1 is an in-circuit debugger/programmer for the STM32 microcontrollers.
- Virtual COM port interface on USB
- Mass storage interface on USB
- Status LED, which blinks during communication with the PC
- USB software re-enumeration
- The debug adaptor supports virtual COM port feature

1. ST-LINK Target SWD Interface

T_JTCK: Clock signal of target CPU, connects to PA14 on STM32
T_JTMS: SWD data input/output, PA13 on the STM32
T_NRST: Reset > NRST on the STM32

2. ST-LINK LED Connection

a) LD1 ST-LINK COM LED

The bicolor LED LD1 (green, red) provides information about ST-LINK communication status.

- *Blinking red*: the first USB enumeration with the PC is taking place
- *Red on*: the initialization between the PC and ST-LINK is complete

- *Blinking red or green*: programming and debugging with target
- *Green on*: communication finished and successful
- *Orange on*: communication failure

b) LD2 PWR

- The red LED indicates that the STM32G0 part is powered and 5V power is available on CN4 pin 4.

c) LD3 USER

The LD3 USER green LED is connected to the following STM32G031K8T6 I/O:
• PB3, if the configuration is SB12 ON, and SB13 OFF
• PC6, if the configuration is SB12 OFF, and SB13 ON (default configuration)

d) LD4 USB power fault (OC, overcurrent)

3. ST-LINK USB Connection

4. ST-LINK SWD INTERFACE

5. Target MCU

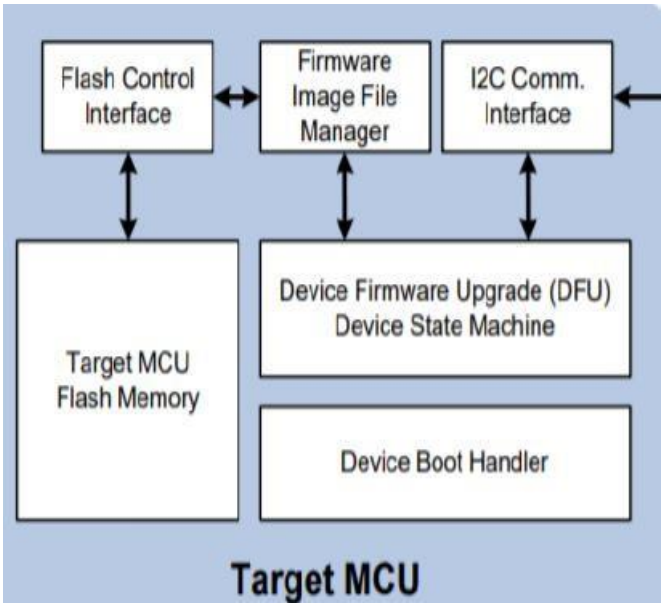

Figure 3. Target MCU

The end device running an I2C bootloader and application firmware. The project MCU target defines the build and debug settings for your target device - compiler settings, memory layout for linker scripts and debug launch configurations. For application projects, you should always match the selected MCU part for the project to the actual target MCU.

6. Extension Connectors

7. ST-LINK MCU

PROBLEM 2

- **LED is connected to the PC6. C represents the port GPIOC, and 6 represents the pin 6.**
- **To turn on an LED, just write a 1 to the corresponding bit location in the ODR register memory address.**

- **Flowchart of PROBLEM 2**

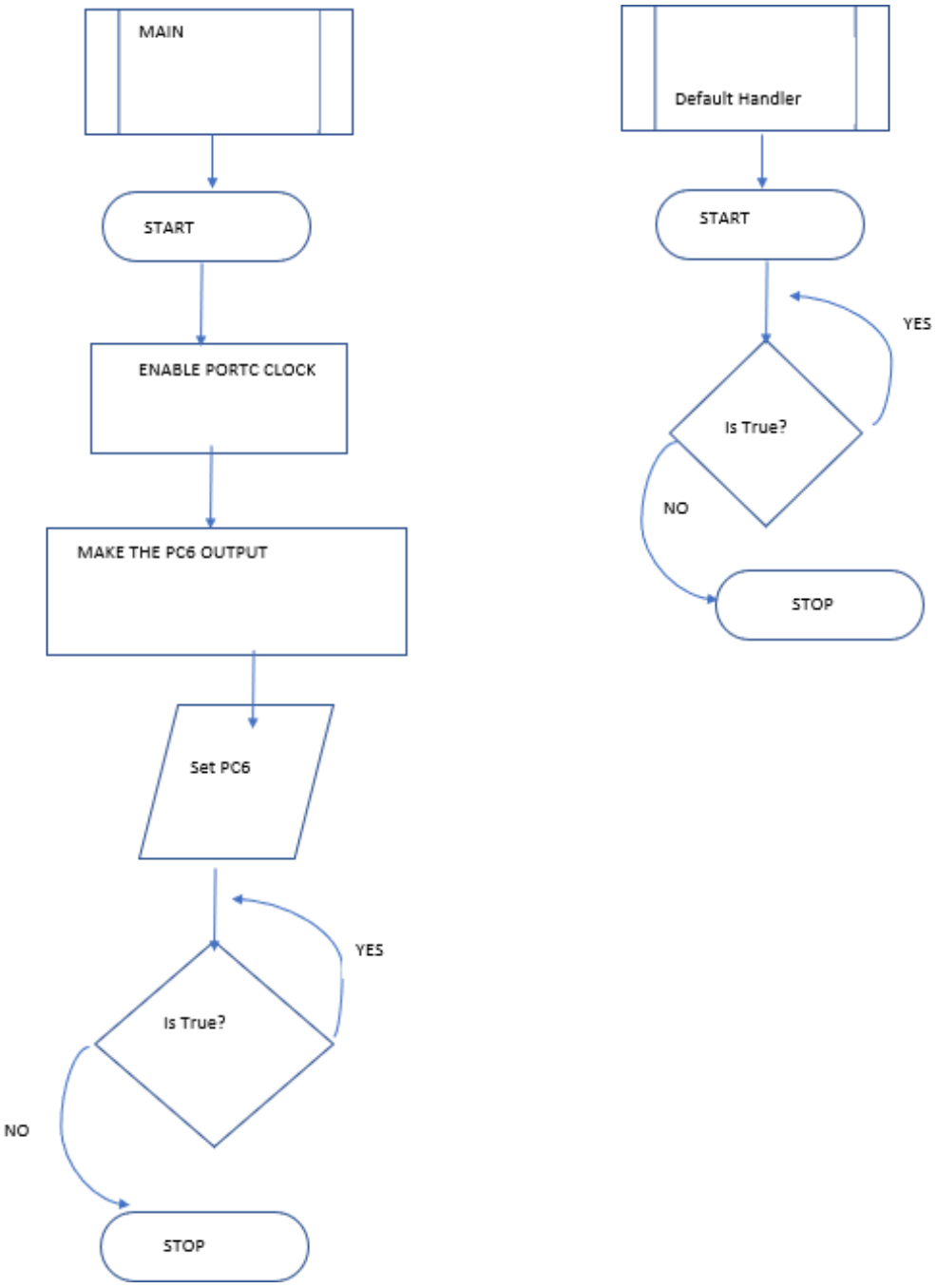

Figure 1.

```
/*PROBLEM 2
 * asm.s
 *
 *
 *
 * description: Added the necessary stuff for turning on the green LED on the
 *              G031K8 Nucleo board.
 */


.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]

	ldr r7,=0x00FFFFFF

delay_led_on:
	//delays 1 second
	subs r7,r7,#1
	bne delay_led_on

	movs r2,#0 //led toggles
	ands r5, r5, r2
	str r5, [r6] //led off

	/* for(;;); */
	b .
	/* this should never get executed */
	nop
```

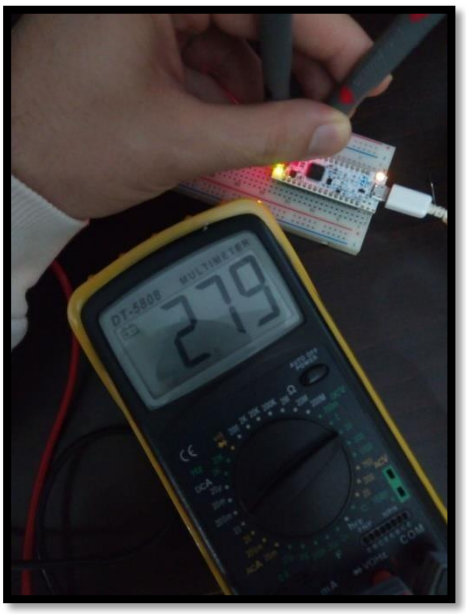

Figure 2.

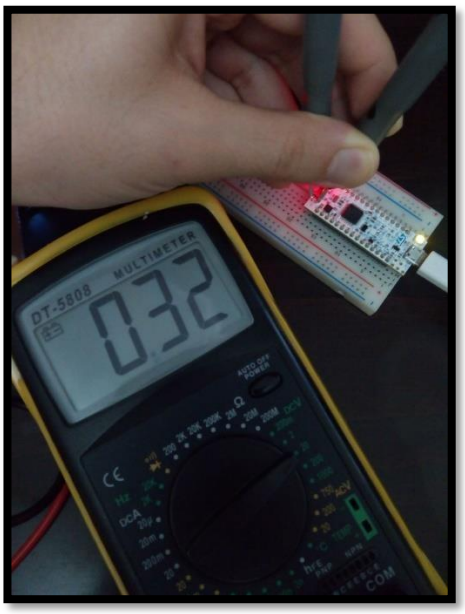

Figure 3.

PROBLEM 3

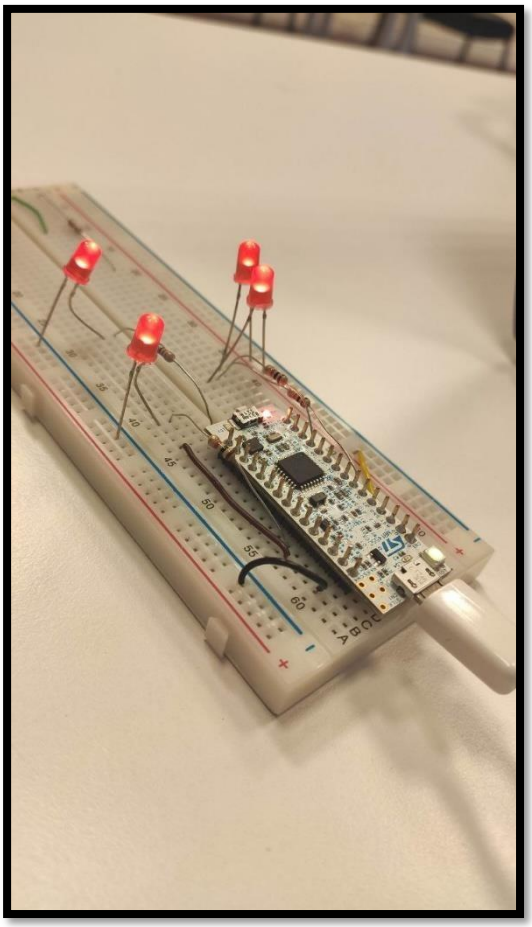

Figure 4.

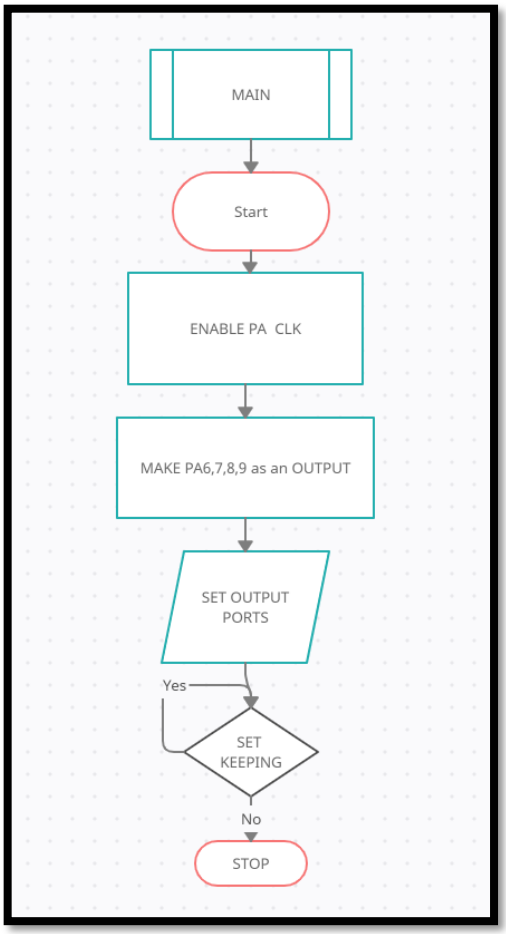

Figure 5.

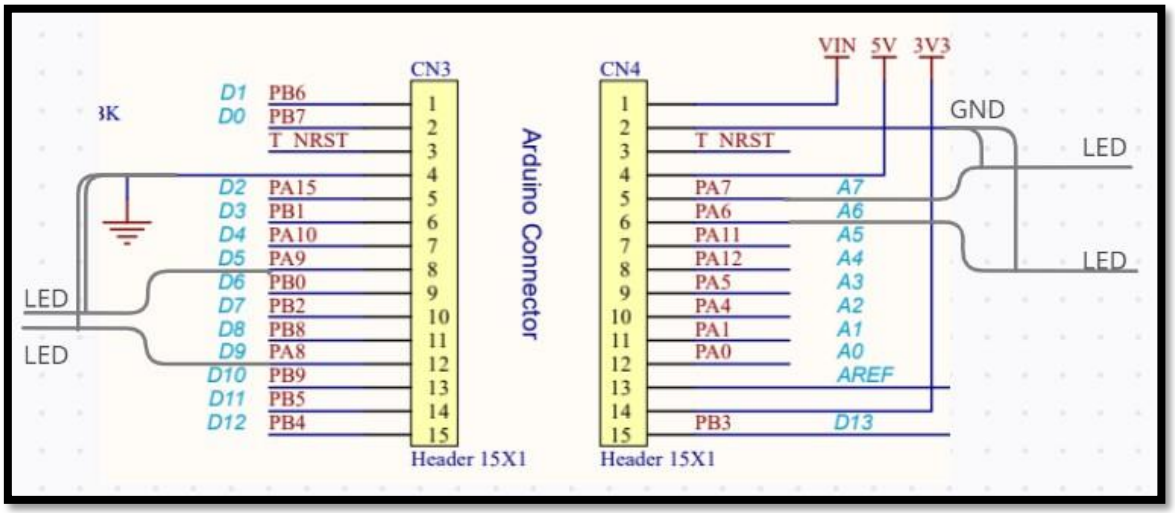

Figure 6.

```
/*PROBLEM3
 * asm.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 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 GPIOA clock, bit0 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]
```

```
/* setup PA6,7,8,9 as an output for leds in MODER */
ldr r6, =GPIOA_MODER
ldr r5, [r6]

ldr r4, =0xFF000
mvns r4, r4                    // r4 = !r4
ands r5, r5, r4                // r5 is clear
ldr r4, =0x55000
orrs r5, r5, r4
str r5, [r6]

/* turn on leds connected to PA6, PA7, PA8, PA9 in ODR */
ldr r6, =GPIOA_ODR
ldr r5, [r6]
ldr r4, =0x3C0
orrs r5, r5, r4
str r5, [r6]

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

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

PROBLEM 4

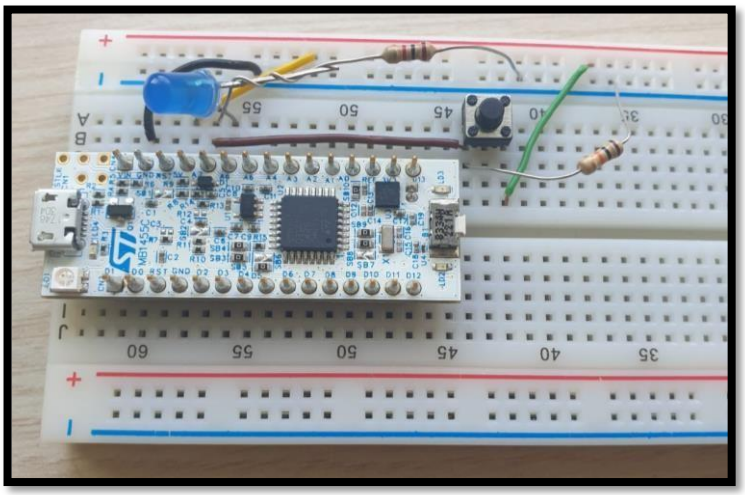

Figure 7.

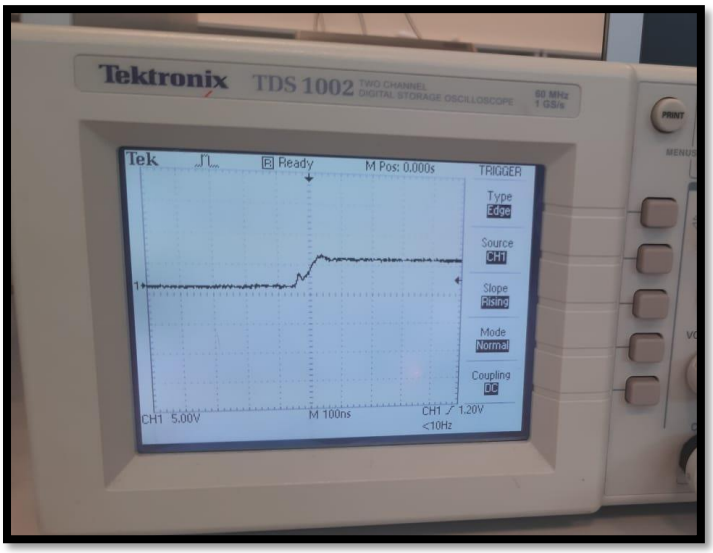

Figure 8.

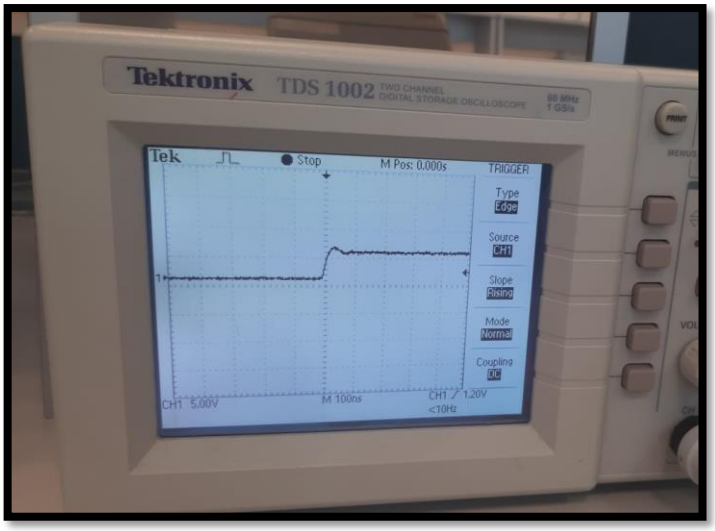

Figure 9.

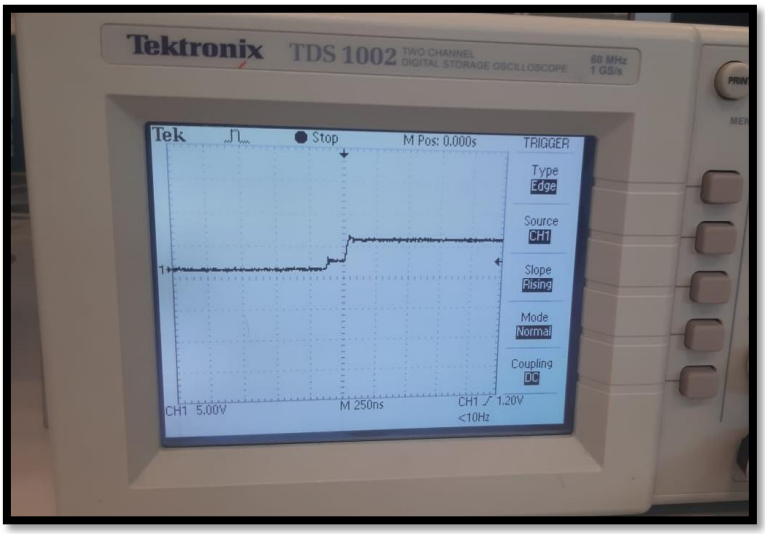

Figure 10.

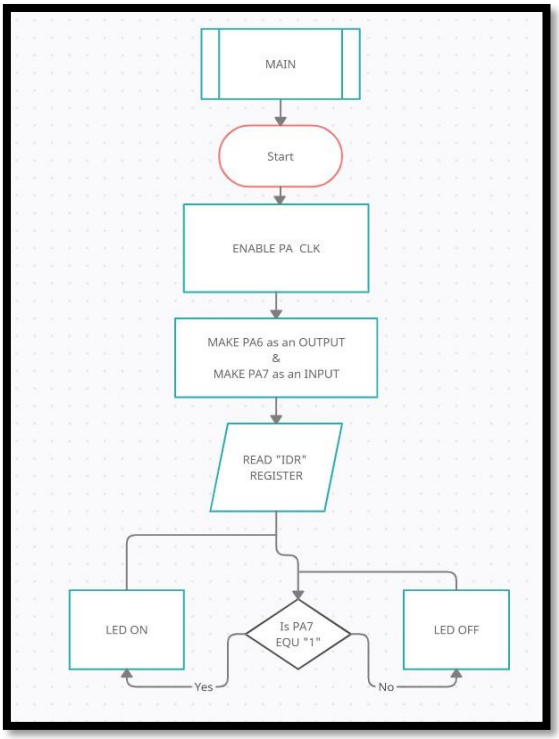

Figure 11.

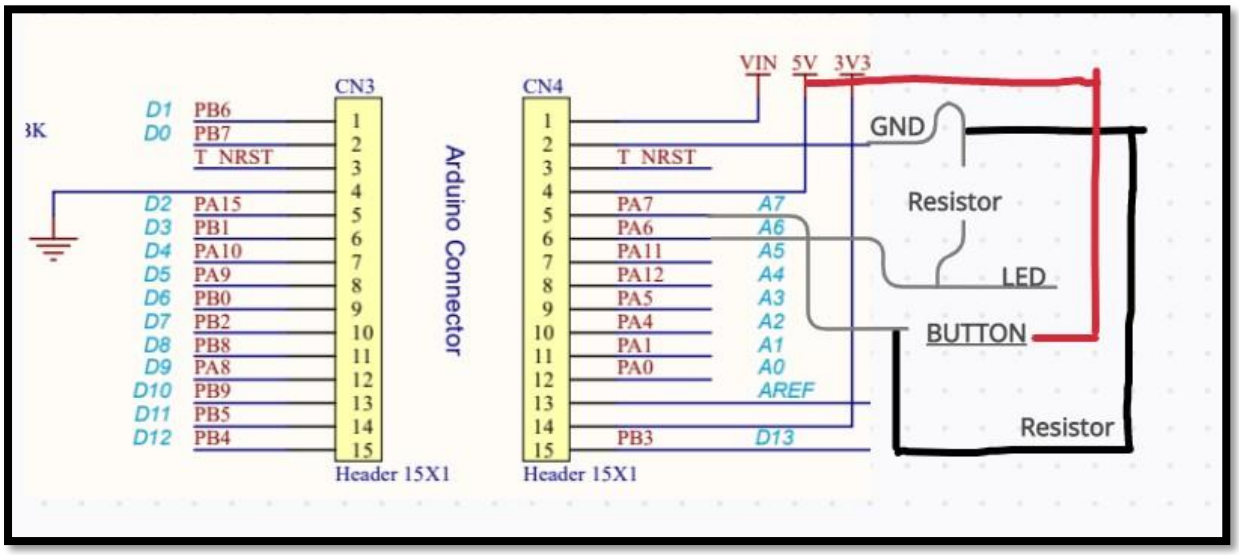

Figure 12.

```
/*
 * Problem4.s
 *
 *
 * Turning On A LED With Button
 * Board that i used is STM32G031.
 * A button and a LED is connected to the board.
```

```
   When the button is pressed LED is on and
   when it's released LED is off immediately.
 */


.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_IDR,        (GPIOA_BASE + (0x10))       // GPIOA IDR 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 GPIOA clock, bit0 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]

/* setup PA6 for led 01 for bits 12-13 and setup PA7 for button for bits 14-15 in MODER */

   ldr r6, =GPIOA_MODER
   ldr r5, [r6]
   ldr r4, =0x3000
   mvns r4, r4
   ands r5, r5, r4
   ldr r4, =0x1000
   orrs r5, r5, r4
   str r5, [r6]    //PA6 output

   ldr r6, =GPIOA_MODER
   ldr r5, [r6]
   ldr r4, =0x3
   lsls r4, r4, #14
   mvns r4, r4
   ands r5, r5, r4
   str r5, [r6] //PA7 input

loop:
    /* Button is pressed*/
    ldr r6, =GPIOA_IDR
    ldr r5, [r6]

    lsrs r5, r5, #7
    movs r4, #0x1
    ands r5, r5, r4

    cmp r5, #0x1 // Button press or not
    beq led_on
    bne led_off

led_on:

    /* turn on led connected to PA6 in ODR */
    ldr r6, =GPIOA_ODR
    ldr r5, [r6]
    movs r4, 0x40
    orrs r5, r5, r4
    str r5, [r6] //Led ON

    b loop // Back to read IDR register

led_off:

   /* turn off led connected to A6 in ODR */
   ldr r6, =GPIOA_ODR
   ldr r5, [r6]
   movs r4, #0x40
```

```
mvns r4, r4
ands r5, r5, r4
str r5, [r6] //Led OFF

 b loop // Back to read IDR register
 nop
```

PROBLEM 5

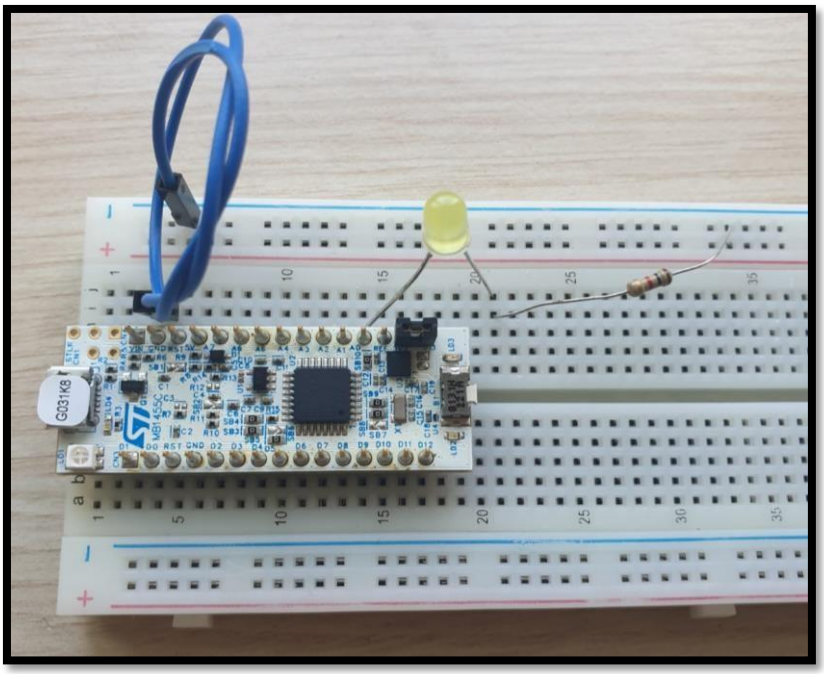

Figure 13.

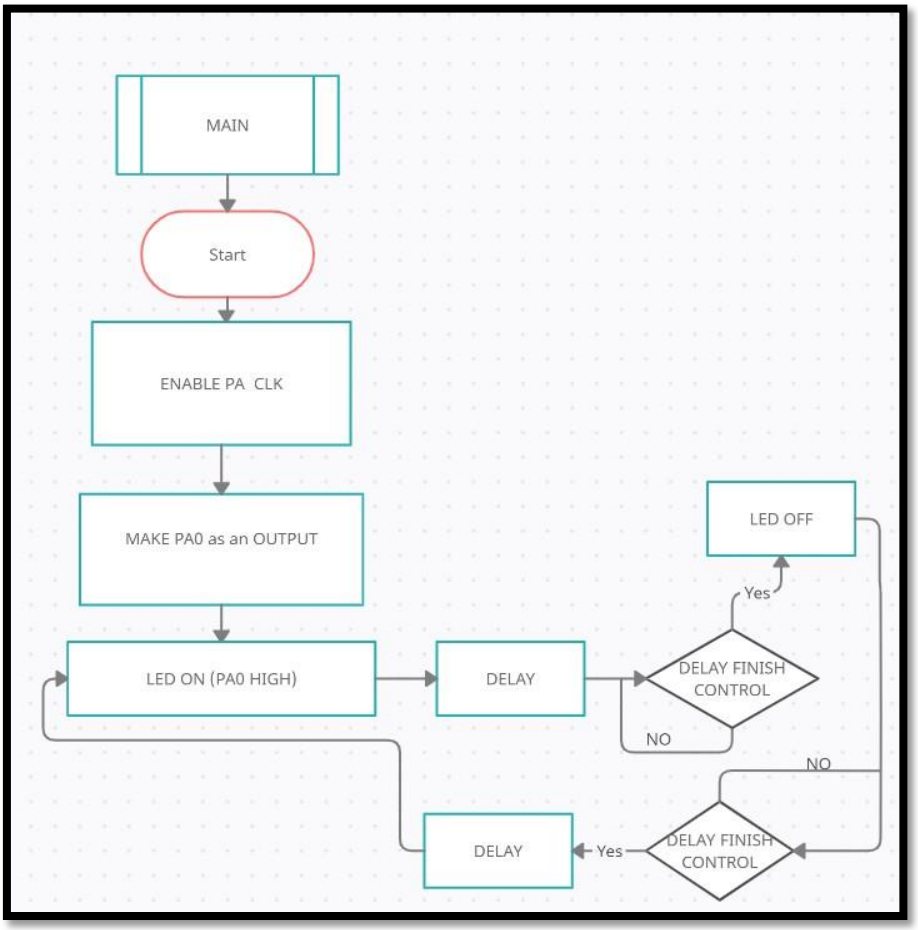

Figure 14.

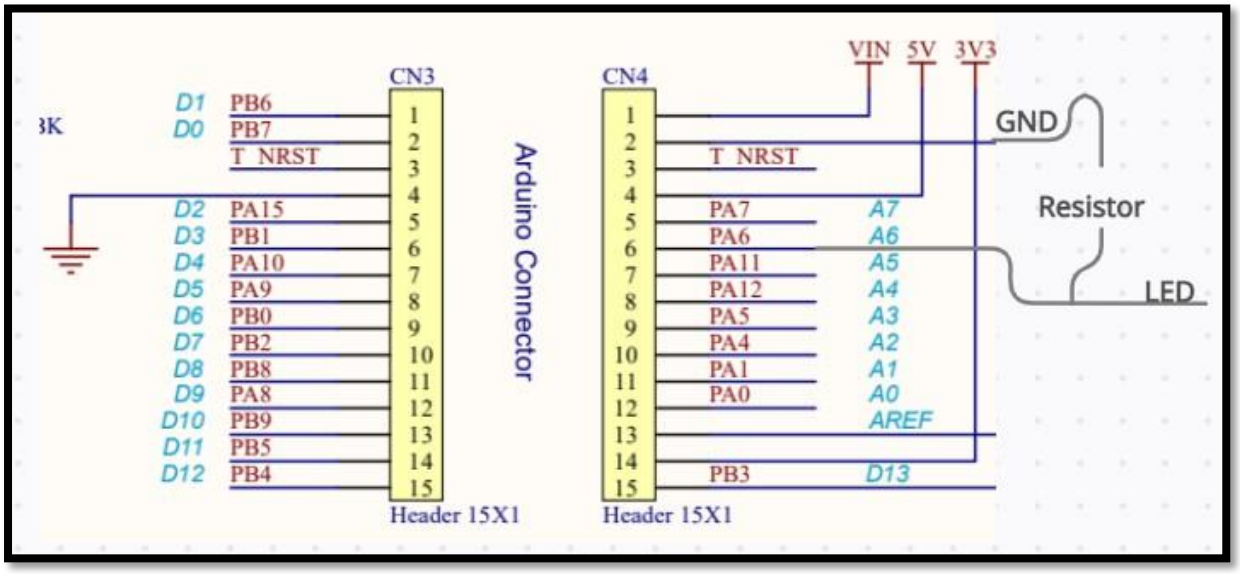

Figure 15.

```
/* PROBLEM 5
 * asm.s
 *
*
 * description: Added the necessary stuff for turning on the yellow external LED by using
 *                  G031K8 Nucleo board.
 */
.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 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 GPIOA clock, bit0 on IOPENR */
        ldr r6, =RCC_IOPENR
        ldr r5, [r6]
        /* movs expects imm8, so this should be fine */
        movs r4, 0x9
        orrs r5, r5, r4
        str r5, [r6]

        /* setup PA0 for led 01 for bits 0-1 in MODER */
        ldr r6, =GPIOA_MODER
        ldr r5, [r6]

        ldr r4, =0xF
        mvns r4, r4             // r4 = !r4
        ands r5, r5, r4         // r5 has been cleaned
        ldr r4, =0x5
        orrs r5, r5, r4
        str r5, [r6]            // PA0 is output

        /* turn on led connected to A0 in ODR */
        ldr r6, =GPIOA_ODR
        ldr r5, [r6]

my_loop:
        ldr r4, =0x1
        orrs r5, r5, r4
        str r5, [r6]    //Led HIGH

        ldr r2, = 0x7A1200 //counter
led_on:
        subs r2, r2, #1
        bne led_on //keep delay

        movs r4,#0
        ands r5,r5,r4
        str r5, [r6]    //Led LOW

        ldr r2, = 0x7A1200 //counter reload
led_off:
        subs r2, r2, #1
        bne led_off //keep delay

        b my_loop //back to “my_loop” line
        b .
```

- **A 16MHz processor can process 1/16000000 instructions in 1 cycle. 1 second must be adjusted. Since SUBS and BNEcommands take 1 cycle, we must repeat 8000000x2 times so that we get 16MHz. That is the reason of 7A1200 assignment.**

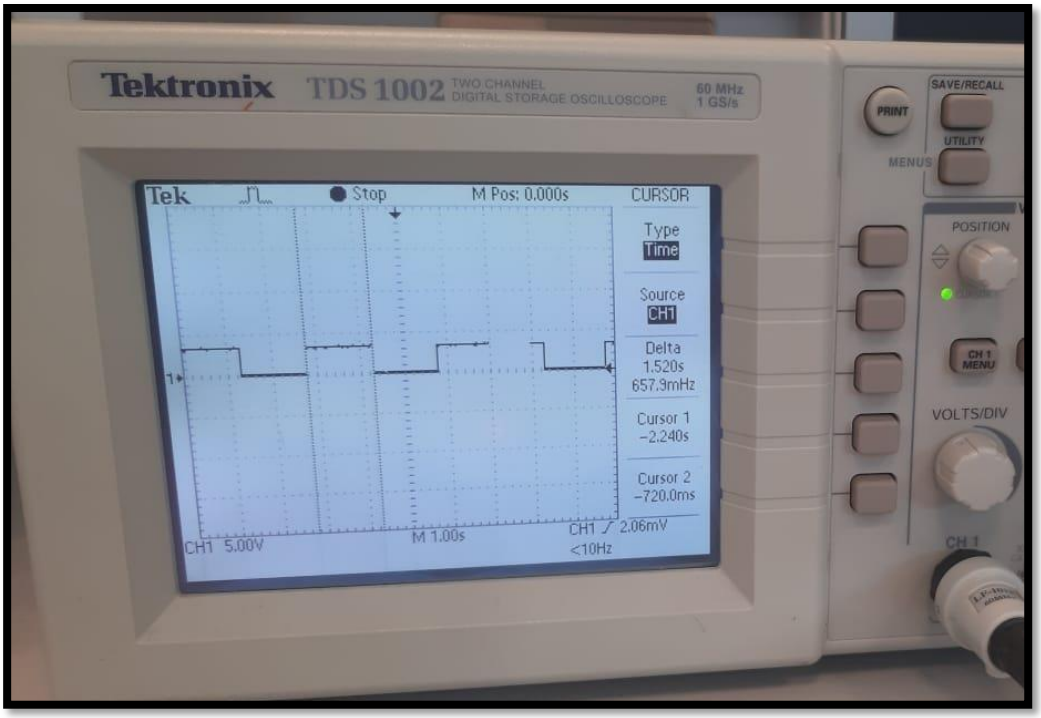

Figure 16.

• LED blinks in each 1.520s periodically.

- **Estimation of CPI**

  - For the multi-cycle MIPS. Load. 5 cycles. Store. 4 cycles. R-type. 4 cycles. Branch. 3 cycles.
  - **Code consists of 55 instructions.**
  - **18 LDR – 32%**
  - **6 STR – 10%**
  - **18 R Type – 32%**
  - **13 Branch Type - 23%**

$$CPI = \frac{32x5 + 10x4 + 32x4 + 23x3}{100} = 3.97$$