ECEN 5803

Mastering Embedded System Architecture

(Fall-2023)

PROJECT-1 : MODULE 2

BUTTON READ, ADC READ, LED PWM, AND UART

Date

10/2/2023

Guided By

Prof. Timothy Scherr

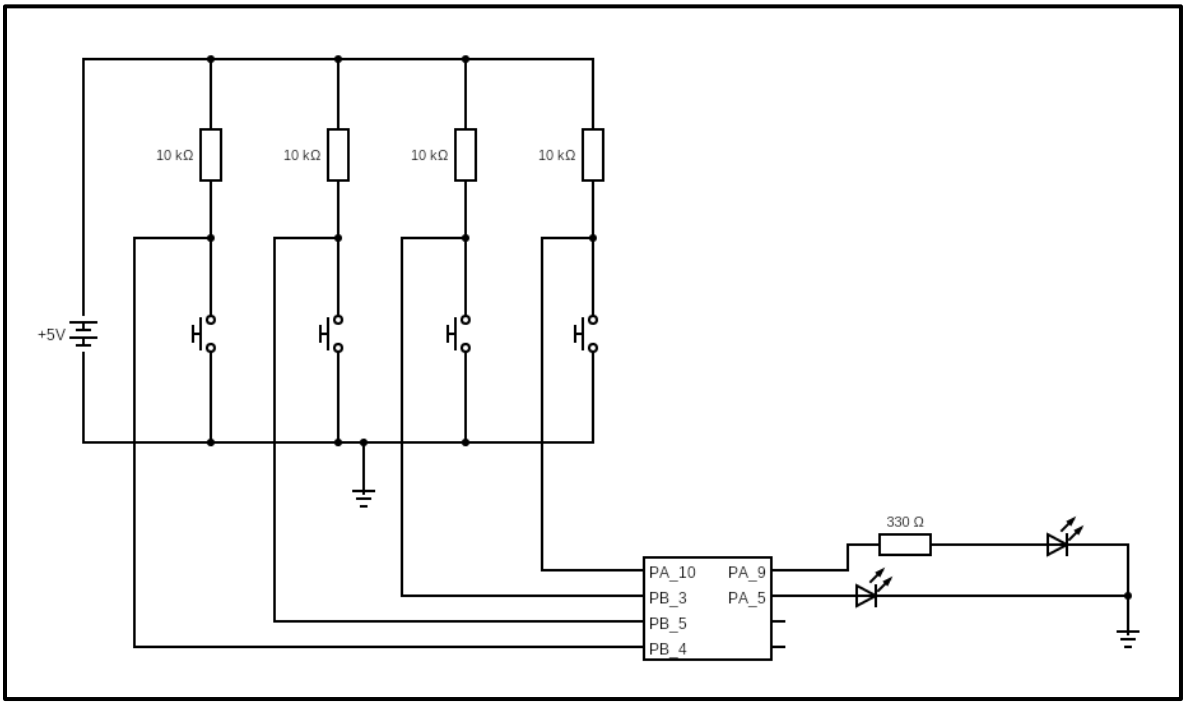
Submitted By

Ajay Kandagal

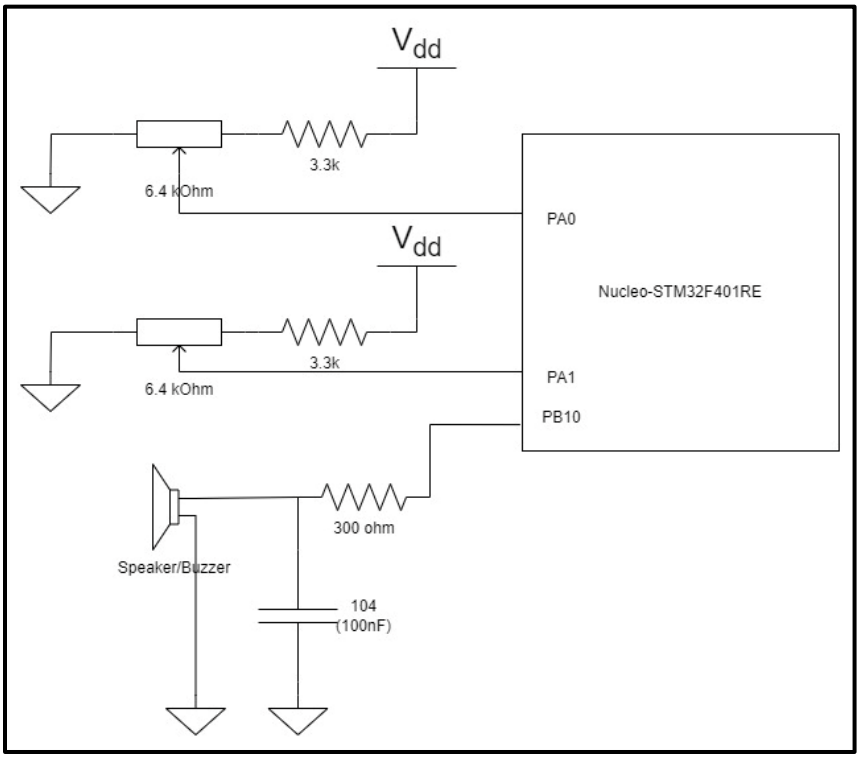
Kshitija Dhondage

Create wiring diagrams:

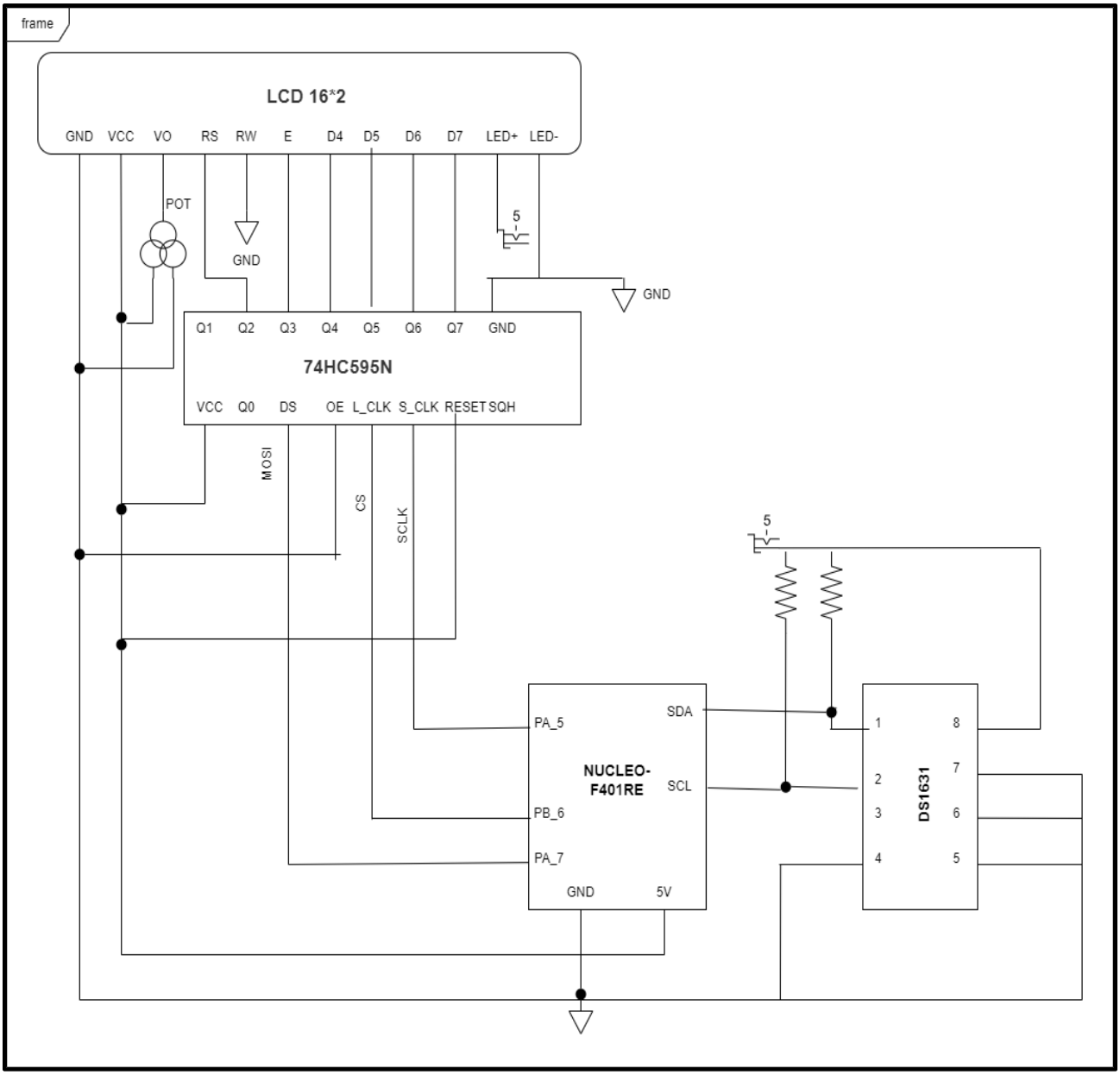
Exercise 2_8.1 & Exercise 2_8.2: Button and LED Interface:



Exercise 2_9: Speaker (Buzzer) interface:



Exercise 2_11. 5: LCD, Temperature Sensor Interface:



Answer the questions in the Lab documents.

Exercise 2_8: Try to issue an interrupt on different signal edges (rising edge or falling edge). What changes?

Ans:

For a rising edge interrupt, the LED state changes when I release the button.

For a falling edge interrupt, the LED state changes when I press the button. (before release).

When configured for a rising edge interrupt, the interrupt will be triggered when the signal transitions from low (0) to high (1). In the context of a button press, this means that the interrupt will occur when we release the button (transition from low to high).

When configured for a falling edge interrupt, the interrupt will be triggered when the signal transitions from high (1) to low (0). In the context of a button press, this means that the interrupt will occur when we press the button (transition from high to low).

Exercise 2_9: What changes when you adjust the amount by which variable i is incremented/decremented?

Ans:

The variable i is used to adjust the duty cycle. The amount by which i is incremented/decremented controls how fast the duty cycle reaches the target value. As a result, it affects the slope of the saw-tooth waveform. If we measure the period of the saw-tooth waveform generated, then the period value decreases as we decrease the amount by which i is changed.

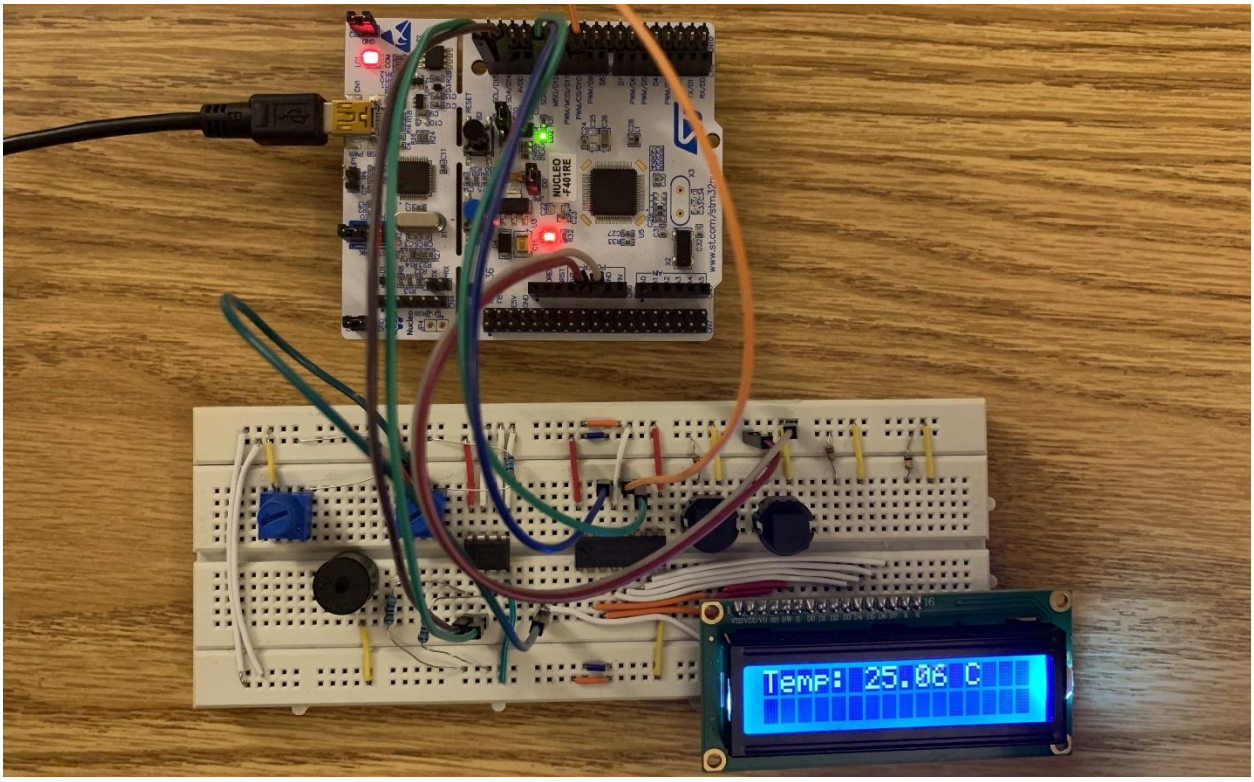
Below is the sawtooth waveform observed across the buzzer when connected to an RC circuit for averaging out the duty cycle. Period: 124 us, Pulse Width: 11 us and Duty Cycle: 0.089.

Exercise 2_11: What temperature is displayed on your PC terminal window? What is displayed on the LCD?

Ans:

The temperature sensor sends temperature data via I2C, which is displayed on the PC terminal through UART and simultaneously on an LCD screen using SPI.

So, the same temperature value is displayed on the LCD and PC terminal.



Appendix

Exercise 2_8.1 :Interface the GPIOs using digital IO functions in the mbed API

Code: digital_io

```
/*
 * @file_name :  main.cpp
 *
 * @authors :  Kshitija Dhondage
 * @Course  :  ECEN-5803 : MESA. Project-1, Module-2
 * @ Date   :  10/1/2023
 *
 * @brief :    This file contains Bus interface for buttons and LED's as input and ouputs.
 *           The code is developed to controlled the LED's by buttons
 *
 */

#include "mbed.h"

/* Output Bus for on board LD2 LED and externel LED connected to PA_9 port */
BusOut My_Bus_Out(LED1, PA_9);

/* Input Bus for four buttons */
BusIn My_Bus_In(PA_10, PB_3, PB_5, PB_4);

/* MACROS definations for buttons */
#define LD2_ON_SWITCH     0x0E     // PA10 1110  14
#define LD2_OFF_SWITCH    0x0D     // PB3  1101  13
#define Ex_LED_ON_SWITCH  0x0B     // PB5  1011  11
#define EX_LED_OFF_SWITCH 0x07     // PB4  0111   7
```

```
/**
 * @brief Main function Entry point function
 *
 * @param[in] Void
 *
 * @return    int
 */
int main()
{
   My_Bus_Out = 0;
   while(1)
   {
      /* button PA_10 press then on board LED will turn ON */
      if(My_Bus_In == LD2_ON_SWITCH)
      {
        My_Bus_Out = (My_Bus_Out | (1 << 0));
      }
      /* button PB_3 press then on board LED will turn OFF */
      if(My_Bus_In == LD2_OFF_SWITCH)
      {
        My_Bus_Out = (My_Bus_Out & ~(1 << 0));
      }
      /* button PB_5 press then off board LED will turn ON */
      if(My_Bus_In == Ex_LED_ON_SWITCH)
      {
        My_Bus_Out = (My_Bus_Out | (1 << 1));
      }
      /* button PB_4 press then off board LED will turn OFF */
```

```
    if(My_Bus_In == EX_LED_OFF_SWITCH)
    {
        My_Bus_Out = (My_Bus_Out & ~(1 << 1));
    }
 }
}
```

Exercise 2_8.2 Implement the interrupt handler using the interrupt libraries in the mbed API

Code: interrupt

```
/*
 * @file_name :  main.cpp
 *
 * @authors :  Kshitija Dhondage.
 * @Course  :  ECEN-5803 : MESA. Project-1, Module-2
 * @ Date   :  10/1/2023
 * @brief   :  This file contains ISR for buttons in which LED indicates when Interrput occurs.
 *
 */


#include "mbed.h"


/* Output Bus for on board LED LD2 and externel LED connected on pin PA_9 */
BusOut My_Bus_Out(LED1, PA_9);


/* interrupts for input button pins  */
InterruptIn button_press_LD_ON(PA_10);
InterruptIn button_press_LD_OFF(PB_3);
InterruptIn button_press_LED_ON(PB_5);
InterruptIn button_press_LED_OFF(PB_4);
```

```
/* ISR for button interrupts */

/* Interrupt handler for button-1 connected on PA_10 */
void button_1_handler()
{
   /* on board LED LD2 turns ON */
   My_Bus_Out = (My_Bus_Out | (1 << 0));
}

/* Interrupt handler for button-2 connected on PB_3 */
void button_2_handler()
{
  /* on board LED LD2 turns OFF */
   My_Bus_Out = (My_Bus_Out & ~(1 << 0));
}

/* Interrupt handler for button-3 connected on PB_5 */
void button_3_handler()
{
   /* Externel LED turns ON */
   My_Bus_Out = (My_Bus_Out | (1 << 1));
}

/* Interrupt handler for button-4 connected on PB_4 */
void button_4_handler()
{
   /* Externel LED turns OFF */
   My_Bus_Out = (My_Bus_Out & ~(1 << 1));
```

```
}

/**
 * @brief Main function Entry point function
 *
 * @param[in] Void
 *
 * @return
 * int
 */
int main(){

   /* Enbale interrupts */
   __enable_irq();

   /* Button Interrupt handlers  */

   /* Attaching the address of the ISR to the rising edge  */
   button_press_LD_ON.rise(&button_1_handler);
   button_press_LD_OFF.rise(&button_2_handler);
   button_press_LED_ON.rise(&button_3_handler);
   button_press_LED_OFF.rise(&button_4_handler);

   /* Attaching the address of the ISR to the falling edge  */
  /*
   button_press_LD_ON.fall(&button_1_handler);
   button_press_LD_OFF.fall(&button_2_handler);
   button_press_LED_ON.fall(&button_3_handler);
   button_press_LED_OFF.fall(&button_4_handler);
```

```
  */

  /* Initially turn off  LED */
  My_Bus_Out = 0;

  /* Sleep on exit */
  while(1)
  {
  __wfi();
  }
}
```

Exercise 2_11 Implement SPI interface for LCD

Code: SPI

```
/*
 * @file_name :  NHD_0216HZ.cpp
 *
 * @authors :  Kshitija Dhondage
 * @Course  :  ECEN-5803 : MESA. Project-1, Module-2
 * @ Date   :  10/1/2023
 *
 * @brief   :  This file contains LCD, SPI initilization.
 *            LCD write command and data functions along with cursor set function.
 *
 */

#include "mbed.h"
#include "NHD_0216HZ.h"
#include "pindef.h"
```

```
/* slave select a.k.a. cs or latch for shift reg */
DigitalOut SS(SPI_CS);

/* Set SPI */
SPI spi(SPI_MOSI, NC, SPI_SCLK);

/* Initialise SPI */
void init_spi(void) {
   SS = 1;

   spi.format(8, 3);        //8bit spi mode 2
   spi.frequency(100000);    //100 kHz spi clock
}

/* Initialise LCD */
void init_lcd(void)
{
  /* Wait for 40 milliseconds */
  wait_ms(40);

  /* Function Set: 8-bit mode, 2-line display, 5x8 font */
  write_cmd(0x30);

  /* Wait for 37 milliseconds */
  wait_ms(37);

  /* Function Set: 8-bit mode, 2-line display, 5x8 font */
  write_cmd(0x20);
```

```
 /* Wait for 37 microseconds  */
 wait_us(37);

 /* Function Set: 8-bit mode, 2-line display, 5x8 font */
 write_cmd(0x20);

 /* Wait for 37 microseconds */
 wait_us(37);

 /* Display On/Off Control: Display on, cursor off, blink off */
 write_cmd(0x0C);

 /* Wait for 4 milliseconds  */
 wait_ms(4);

 /* Clear Display */
 write_cmd(0x01);

 /* Wait for 4 milliseconds  */
 wait_ms(4);

 /* Entry Mode Set: Increment cursor position, no display shift */
 write_cmd(0x06);

 /* Wait for 4 milliseconds  */
 wait_ms(4);
}
```

Exercise 2_11 Implement UART interface

Code: UART

```
/*
 * @file_name :  main.cpp
 *
 * @authors :  Kshitija Dhondage
 * @Course  :  ECEN-5803 : MESA. Project-1, Module-2
 * @ Date   :  10/1/2023
 *
 * @brief :    This file contains UART interface from board to PC
 *
 *
 */
#include "mbed.h"
#include "pindef.h"

/* Serial tx, rx connected to the PC via an USB cable */
Serial device( PA_2, PA_3 );

/**
 * @brief      Function to set baudrate for UART serial port
 *
 * @param[in]  Baudrate value
 *
 * @return     Void
 */

void baudrate_set(int baudrate)
{
```

```
   /* set serial port (UART) baudrate */
    device.baud(baudrate);
}

/**
 * @brief Main function Entry point function
 *
 * @param[in] Void
 *
 * @return    int
 */
int main()
{

   /* Call the function to set the baudrate */
   baudrate_set(9600);

  while(1)
  {
    device.printf("\n\r MESA Project-1 Module-2 \n\r"); // Print "MESA Project-1 Module-2" to the PC serial monitor
  }
 }
```

Exercise 2_11 Implement I2C interface for DS1631

Code: I2C

```
/*
 * @file_name :  main.cpp
 *
 * @authors :  Kshitija Dhondage
```

```
 * @Course  :  ECEN-5803 : MESA. Project-1, Module-2
 * @ Date   :  10/1/2023
 *
 * @brief :    This file contains I2C interface for Temperature Sensor. It contains the code
 *           which reads the temp from sensor via I2C interface and prints on PC terminal via UART
 *
 *
 */

#include "mbed.h"
#include "pindef.h"

/* Serial connection via UART for PC terminal*/
Serial pc(UART_TX, UART_RX);

/* I2C interface for temp sensor: SDA and SCLK pins */
I2C temp_sensor(PB_9,PB_8);

/* I2C address of temperature sensor DS1631 */
const int temp_addr = 0x90;

/* commad array for DS1631 */
char cmd[] = {0x51, 0xAA};

/*Buffer to store temp data read */
char read_temp[2];

/**
 * @brief Main function Entry point function
```

```
 *
 * @param[in] Void
 *
 * @return    int
 */
int main()
{
   while(1)
   {
      /* Write the "Start Convert T" command to the sensor which Initiates temperature conversions */
      temp_sensor.write(temp_addr, &cmd[0], 1);

      wait(0.5);

      /* Write the "Read Temperature command" to the sensor which
     Reads the last converted temperature value from the 2-byte temperature register */
      temp_sensor.write(temp_addr, &cmd[1], 1);

      /* Read the 16-bit temperature data */
      temp_sensor.read(temp_addr, read_temp, 2); // read the temp

      /* Convert  captured temperature from sensor via I2C interface to Celsius */
      float temp = (float((read_temp[0] << 8) | read_temp[1]) / 256);

      /* Temperature printing to the serial monitor */
     pc.printf("\n\r Temp= %f Celcius \n\r", temp);
   }
}
```

Exercise 2_11 Implement code to display temperature captured from DS1631 on LCD

And PC terminal

Code: Integration

```
/*
 * @file_name :  main.cpp
 *
 * @authors :  Kshitija Dhondage
 * @Course  :  ECEN-5803 : MESA. Project-1, Module-2
 * @ Date   :  10/1/2023
 *
 * @brief :    This file contains the code to display the temperature captured form DS1631 via I2C interface
 *           on LCD connected to Nucleo board via SPI interface and on PC terminal via UART interface
 *
 *
 *
 */

#include "NHD_0216HZ.h"
#include "DS1631.h"
#include "pindef.h"

/* Serial UART interface */
Serial pc(SERIAL_TX, SERIAL_RX);

/* variable to store temperature measurement */
float temp;

/* Set the temp sensor DS1631 , I2c_SDA, I2c_SCL, and address */
DS1631 ds1631(PB_9,PB_8,0x90);
```

```
/* Set LCD parameters: CS, MOSI, SCL for SPI interface */
NHD_0216HZ nhd_0216hz(PB_6,PA_7,PA_5);



/**
 * @brief Main function Entry point function
 *
 * @param[in] Void
 *
 * @return    int
 */
int main()
{

  /* LCD Initilization */
    nhd_0216hz.init_lcd();

    while(1)
    {
       /* set the LCD cursor */
       nhd_0216hz.set_cursor(0,0);

       /* Read the temperature */
       temp = ds1631.read();

       /* Display the temp on serially connected PC terminal */
       pc.printf("\n\r Temp: %0.2f C \n\r", temp);
```

```
      /* Display the temp on LCD via SPI */
      nhd_0216hz.printf("Temp: %0.2f C",temp);

      wait_ms(40);
   }
}
```

Exercise 2_9 Analog Input and Output

```
/*****************************************************************************
 * @file    main.c
 * @brief   Code for Project 1 Module 2: Analog Input and Output
 * @version v1.00
 * @date    2, October, 2023
 * @author  Ajay Kandagal (ajka9053@colorado.edu)
 *
 * @details The code uses calls for reading two potentiometers which are used
 *        for controlling the pitch and volume of buzzer driven via PWM
 *        generated by the MCU.
 *****************************************************************************/

#include "mbed.h"
#include "pindef.h"

// Pin definitions
#define BUZZER_OUT    PB_10
#define PULSE_IN                PA_0
#define PERIOD_IN               PA_1

// Frequency range audible by humans
```

```
#define MAX_FREQ                8000U
#define MIN_FREQ                320U
#define FREQ_RANGE  (MAX_FREQ - MIN_FREQ)

// Potentiometer 16-bit read range
#define MAX_POT_VAL 4095
#define MIN_POT_VAL  5
#define POT_RANGE   (MAX_POT_VAL - MIN_POT_VAL)

// Sawtooth control parameters
#define STEPS_COUNT 25
#define STEP_DELAY  5

#define SEC_TO_USEC   1000000

//Define variables
Serial serial_pc(SERIAL_TX, SERIAL_RX);
PwmOut buzzer_obj(BUZZER_OUT);
AnalogIn pulse_obj(PULSE_IN);
AnalogIn period_obj(PERIOD_IN);

/**************************************************************************
 * @brief Contains calls for reading the potentiometers and then controlling a
 *       buzzer via PWM.
 *
 * @details Adjusts the pitch by reading the raw 12-bit value of the potentiometer
 *        and volume using another potentiometer. The period and the duty cycle
 *        are calculated using these raw values, which are then used to drive
 *        the buzzer by generating PWM signal.
```

```
 *
 ******************************************************************************/
int main(){
        float duty_val, max_duty_cycle;
        float step_size;
        uint16_t new_freq;
        uint16_t period_ms;
        uint16_t pulse_ms;
        uint16_t freq_raw;
        uint16_t volume_raw;

        while(1){
                /*
                Print values of the potentiometers to the PC serial terminal
                Create a saw-tooth sound wave
                Make the period and volume adjustable using the potentiometers
                */
                freq_raw = 0;
                volume_raw = 0;

                // Read the potentiometers to adjust pitch and volume
                freq_raw = period_obj.read_u16();
                volume_raw = pulse_obj.read_u16();

                // Map the raw potentiometer value to predefined frequency range
                new_freq = MIN_FREQ + (((float)freq_raw * FREQ_RANGE) / POT_RANGE);

                // Calculate period using frequency expressed in micro-seconds
                period_ms = SEC_TO_USEC / new_freq;
```

```
                // Calcualte pulse-on time
                pulse_ms = (period_ms * volume_raw) / POT_RANGE;

                max_duty_cycle = (float)pulse_ms / period_ms;

                // Calculate number of steps required to generate saw-tooth waveform
                step_size = max_duty_cycle / STEPS_COUNT;

                serial_pc.printf("Pitch raw: %d\n\r", freq_raw);
                serial_pc.printf("Volume raw: %d\n\r", volume_raw);
                serial_pc.printf("\n\r####################\n\r");

                duty_val = duty_val + step_size;

                if (duty_val > max_duty_cycle)
                        duty_val = 0.0;

                // Write PWM values to buzzer
                buzzer_obj.period_us(period_ms);
                buzzer_obj.write(duty_val);

                wait_ms(STEP_DELAY);
        }
}
```