University of Colorado **Boulder**

Viraj Patel & Kiran Jojare
Project 1 Module 2
ECEN 5803 MESA

# Exercise2_8 : Programming GPIO Using mbed api

## Breadboard Connections :

Please find the final breadboard connections where we implement the Exercise 2.8 requirements. The kit given to us only had 3 switches , so a spare switch was obtained and used as seen below in figure (1).

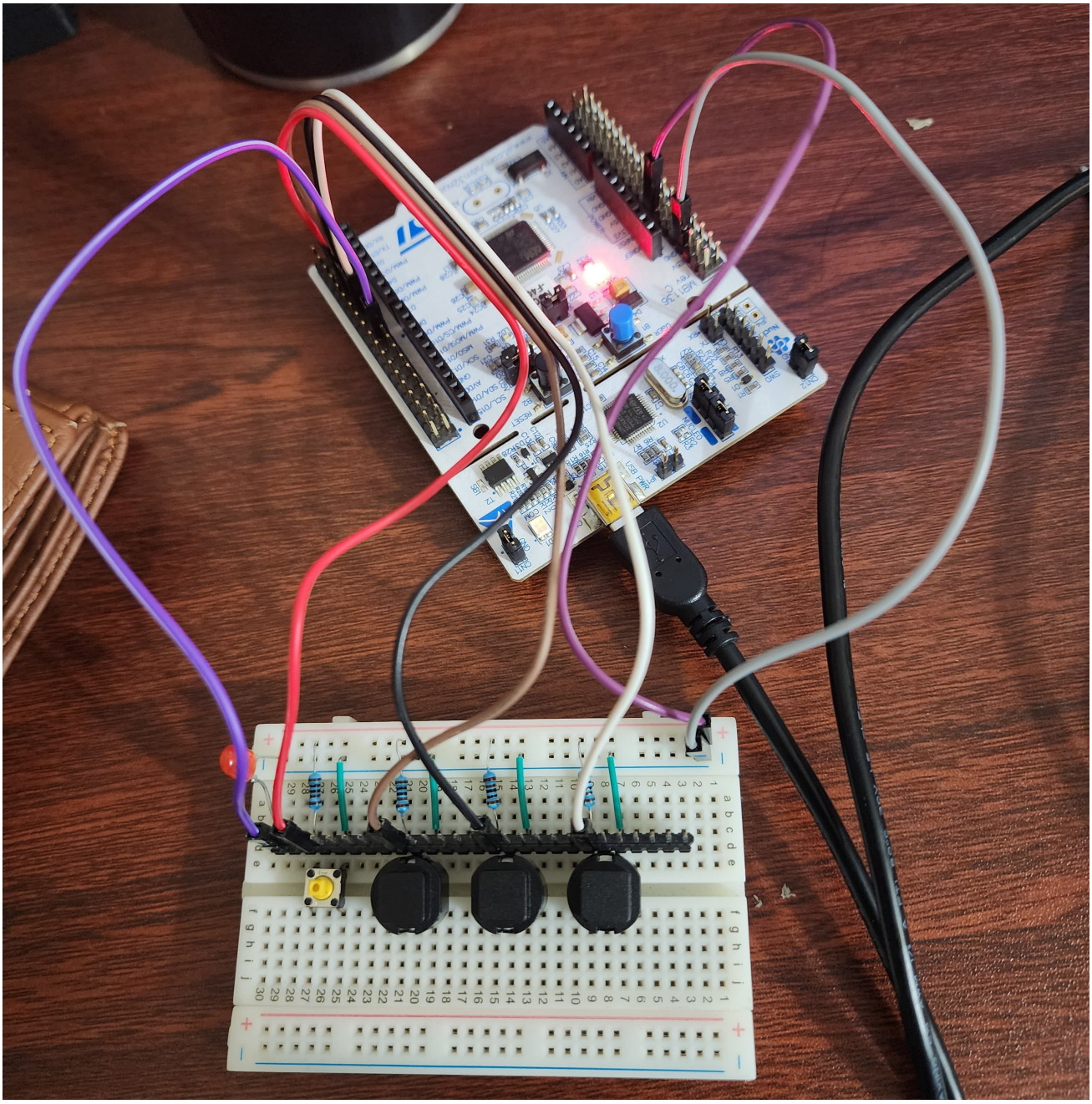

*Figure 1 : Breadboard connection Lab Exercise 2_8*

## Wiring Diagram :

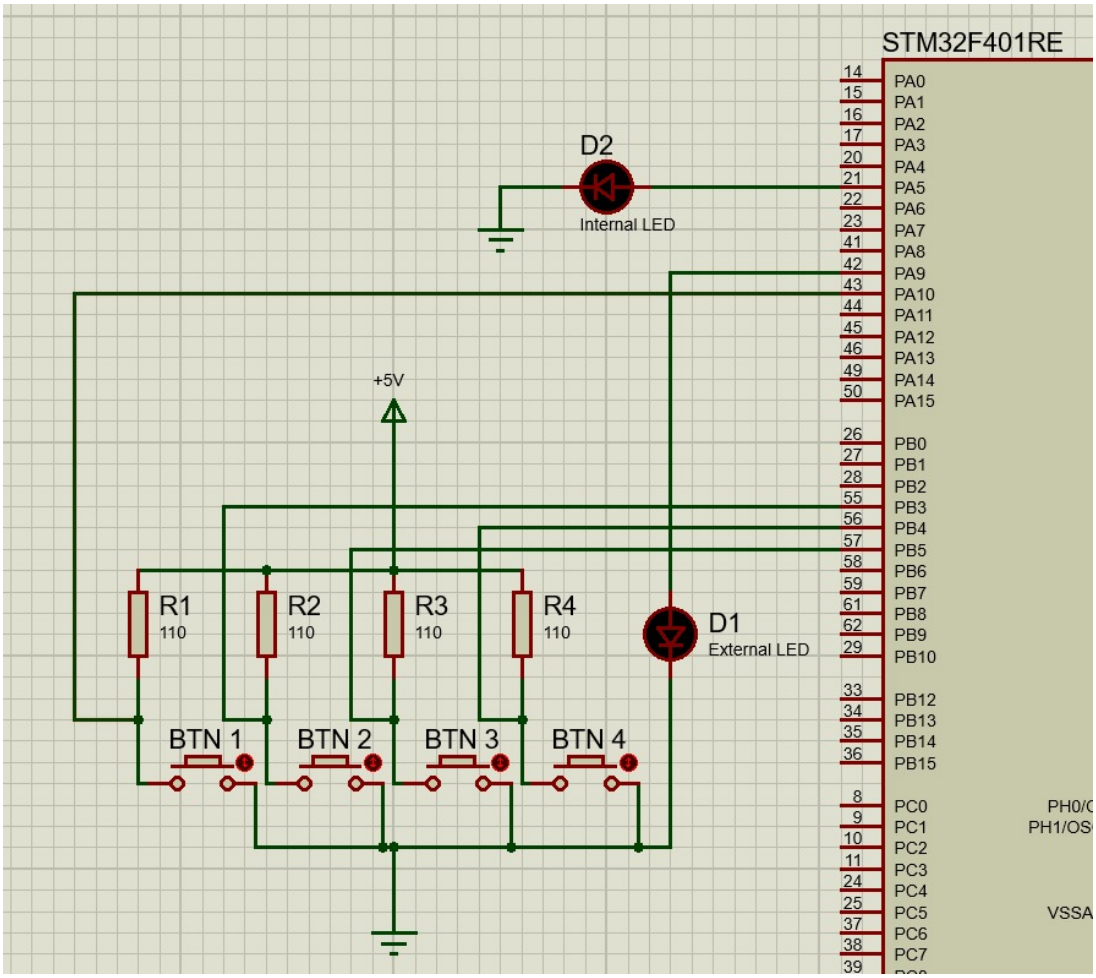


*Figure 2 : Proteus 8 based Wiring Diagram for Exercise 2_8*

## Exercise 1 : Interface the GPIOs using digital IO functions in the mbed API without interrupts

For this exercise we implemented BusIn, BusOut interfaces for inputs and outputs. The working of breadboard is as expected which is as follows. Refer the GIF attached below for the working of our implementation.

- Button 1 turn internal LED LD2 on
- Button 2 turns it off
- Button 3 turns on an external LED on.
- Button 4 turns the external LED off

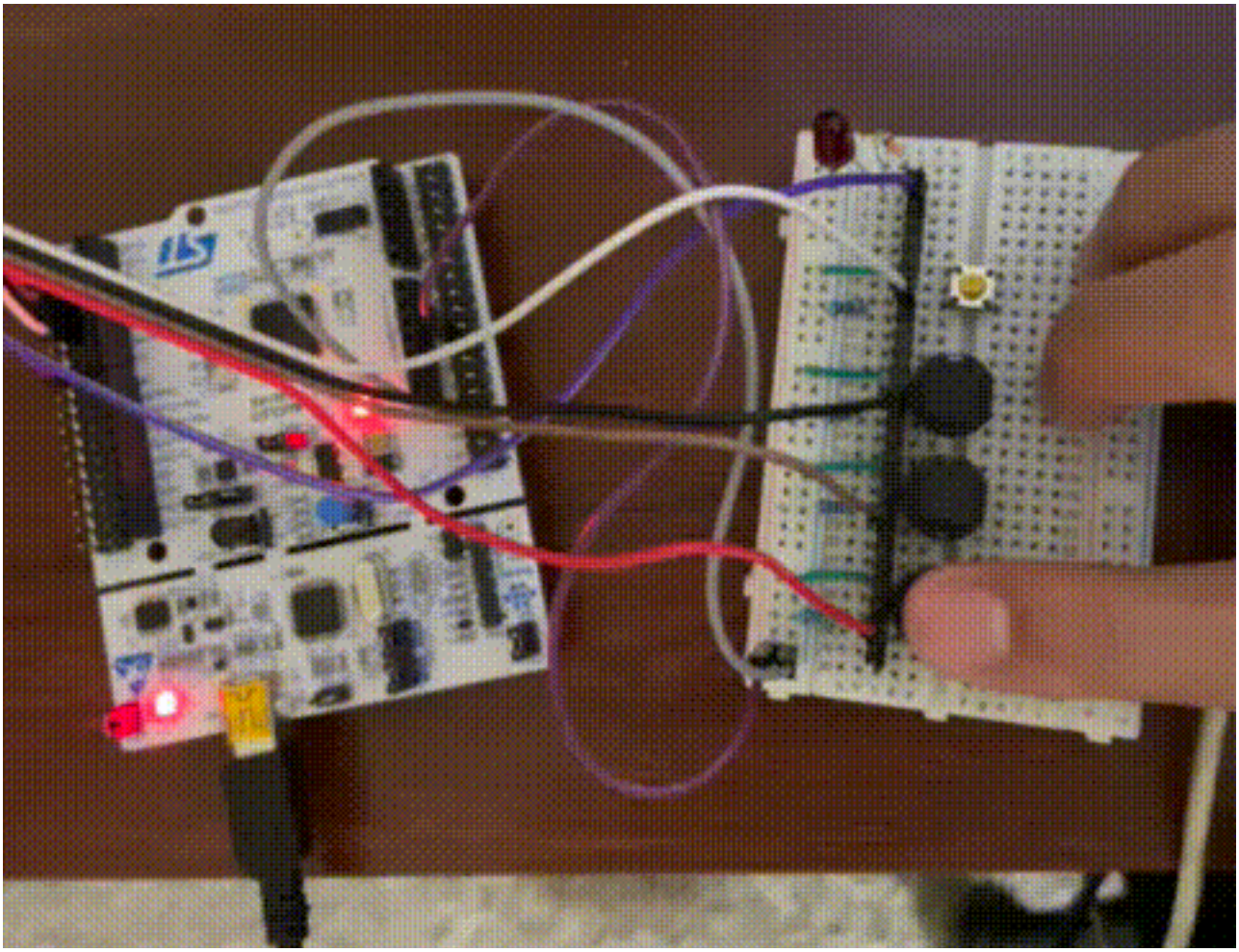

*Figure 3 : Working Exercise Exercise 1*

## *Code :*

```
/*----------------------------------------------------------------------------
LAB EXERCISE 8.1 - DIGITAL INPUTS AND OUTPUTS
PROGRAMMING USING MBED API
 ----------------------------------------
 In this exercise you need to use the mbed API functions to:

       1) Define BusIn, BusOut interfaces for inputs and outputs
       2) The LED is controlled by the button:
                     + USER_BUTTON - LED1

       GOOD LUCK!
 *----------------------------------------------------------------------------*/
/*****************************************************************************
* Copyright (C) 2023 by Kiran Jojare & Viraj Patel
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Kiran Jojare & Viraj Patel and the University of Colorado are not
* liable for any misuse of this material.
*
*****************************************************************************/

/**
* @file_name    main.cpp
* @introduction This file illustrates the use of digital inputs and outputs
*               using the mbed API in an ARM-based microcontroller, focusing
*               on interfacing and controlling LEDs with buttons.
*
```

```
* @author       Kiran Jojare, Viraj Patel
* @date         Sep 30 2023
* @version      1.0
*/

#include "mbed.h"

// Define input bus
BusIn buttons(PB_4, PB_5, PB_3, PA_10); // Define a bus of input pins connected to buttons.

// Define output bus for the LED
BusOut leds(LED2, PA_9);                                                    // Define a bus of
output pins connected to LEDs.

/**
 * @brief Main function initializes LEDs and implements the main program loop.
 * @details In the infinite while loop, the program continuously checks the state of the buttons
 *          and turns on/off corresponding LEDs based on the button pressed.
 */
int main(){
    leds = 0x00; // Initialize LEDs to OFF

    while(1) {
        // Check the state of buttons and actuate corresponding LEDs.

        // if PA_10 is pressed
        if(buttons == 0b00000111) {
            leds = leds | 0b00000001; // Turn internal LED ON
        }

        // if PB_3 is pressed
        if(buttons == 0b00001011) {
            leds = leds & 0b00000000; // Turn internal LED OFF
        }

        // if PB_5 is pressed
        if(buttons == 0b00001101) {
            leds = leds | 0b00000010; // Turn external LED ON
        }

        // if PB_4 is pressed
        if(buttons == 0b00001110) {
            leds = leds & 0x00;                     // Turn internal LED OFF
        }
    }
}

// ********************************ARM University Program Copyright (c) ARM Ltd
2014*************************************
```

## Exercise 2 : Implement the interrupt handler using the interrupt libraries in the mbed API

This part utilizes the interrupt based embed API to confirm the same functionality as seen in figure (3).

**Note** : The code was tested for both rising edge as well as falling edge based interrupt functionality. Code submitted with rising edge. To test it with falling edge comment the rising edge section and uncomment the falling edge section in the code attached below. Rest of the code should remain the same.

### *Code :*

```
/*-----------------------------------------------------------------------------
LAB EXERCISE 8.2 - INTERRUPT IN/OUT
PROGRAMMING USING MBED API
 ---------------------------------------
 In this exercise you need to use the mbed API functions to:

       1) Define InterruptIn and ISR for the button press
       2) In the interrupt service routine, the LED is used to indicate when a
       button was pressed:
                     + USER_BUTTON - LED1

       3) Put the processor into sleep mode upon exiting from the ISR

       GOOD LUCK!
 *-----------------------------------------------------------------------------*/

/*****************************************************************************
* Copyright (C) 2023 by Kiran Jojare & Viraj Patel
*
* Redistribution, modification or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Kiran Jojare & Viraj Patel and the University of Colorado are not liable for
* any misuse of this material.
*
*****************************************************************************/

/**
* @file_name    main.cpp
* @introduction This file demonstrates the use of the mbed API for handling
*               digital inputs and outputs using interrupts in an ARM-based
*               microcontroller, focusing on controlling LEDs with button presses.
*               -- Note -- : This implementation is using rising interrupt
*                            Falling edge based interrupt implementation is commented out
*
* @author       Kiran Jojare & Viraj Patel
* @date         Sep 30 2023
* @version      1.0
*/

#include "mbed.h"

// Define BusOut for LEDs to control them as per button interrupts
BusOut leds(LED2, PA_9);

// Define InterruptIn objects for each button to handle interrupts
InterruptIn btnInternalOn(PA_10);  // Internal ON button
InterruptIn btnInternalOff(PB_3);  // Internal OFF button
InterruptIn btnExternalOn(PB_5);   // External ON button
InterruptIn btnExternalOff(PB_4);  // External OFF button


// Define ISRs for the interrupts
// ISR to handle Internal LED On button press
/**
```

```
 * @brief Handle Internal LED On button press.
 * @details This ISR sets the bit corresponding to the internal LED to 1 (ON)
 * when the button is pressed.
 */
void handleInternalOn() {
    leds = leds | 0b00000001; // Turn internal LED ON
}
/**
 * @brief Handle Internal LED Off button press.
 * @details This ISR resets all bits to 0 (OFF) when the button is pressed.
 */
void handleInternalOff() {
    leds = leds & 0b00000000; // Turn internal LED OFF
}
/**
 * @brief Handle External LED On button press.
 * @details This ISR sets the bit corresponding to the external LED to 1 (ON)
 * when the button is pressed.
 */
void handleExternalOn() {
    leds = leds | 0b00000010; // Turn external LED ON
}
/**
 * @brief Handle External LED Off button press.
 * @details This ISR resets all bits to 0 (OFF) when the button is pressed.
 */
void handleExternalOff() {
    leds = leds & 0x00; // Turn external LED OFF
}
/*-------------------------------------------------------------------------
 MAIN function
 *-------------------------------------------------------------------------*/
/**
 * @brief Main Function.
 * @details The main function initializes the LEDs, enables global interrupts,
 *          and attaches the address of the ISR to the rising edge of the button's signal.
 *          It then enters an infinite while loop where the processor is put to sleep
 *          and waits for an interrupt to occur.
 */

int main(){

        __enable_irq();                     //enable interrupts

        //initially turn off  LED
        //Write your code here
        leds = 0x00; // Initialize LEDs to OFF

        //Interrupt handlers
        //Attach the address of the ISR to the rising edge

        // Attach the address of the ISR to the rising edge of the button's signal
        btnInternalOn.rise(&handleInternalOn);
        btnInternalOff.rise(&handleInternalOff);
        btnExternalOn.rise(&handleExternalOn);
        btnExternalOff.rise(&handleExternalOff);

        // Commented out the falling edge interrupt calls and was also tested

        // btnInternalOn.fall(&handleInternalOn);
        // btnInternalOff.fall(&handleInternalOff);
        // btnExternalOn.fall(&handleExternalOn);
        // btnExternalOff.fall(&handleExternalOff);

        // Sleep on exit
        // The main thread remains in sleep mode, waiting for interrupts to wake it up
        while(1) {
                        __wfi();  // Enter sleep mode and wait for an interrupt to occur
        }
}

// ********************************ARM University Program Copyright (c) ARM Ltd
2014*************************************
```

## Lab Questions :

### Question 1: Try to issue an interrupt on different signal edges (rising edge or falling edge). What changes

We implemented interrupts using the mbed API. We configured interrupts to be triggered on both the rising and the falling edges by modifying the interrupt configurations in the code. The microcontroller was interfaced with LEDs and buttons, and the changes in LED states were observed to understand the behaviour of interrupts.

```
// For rising edge
btnInternalOn.rise(&handleInternalOn);

// For falling edge
// btnInternalOn.fall(&handleInternalOn);
```

In our experiment, we attentively observed the behaviour of the LEDs when interrupts were triggered on the rising edge and the falling edge. Surprisingly, we didn't notice any visible changes or differences in the LED behaviours between the two configurations. The LEDs responded similarly in both cases, with no discernible differences in response times or behaviour.

# Exercise 2_9 : Analog input and output

## Breadboard Connections :

Please find the final breadboard connections we implemented to fulfil the Exercise 2.9 requirements.

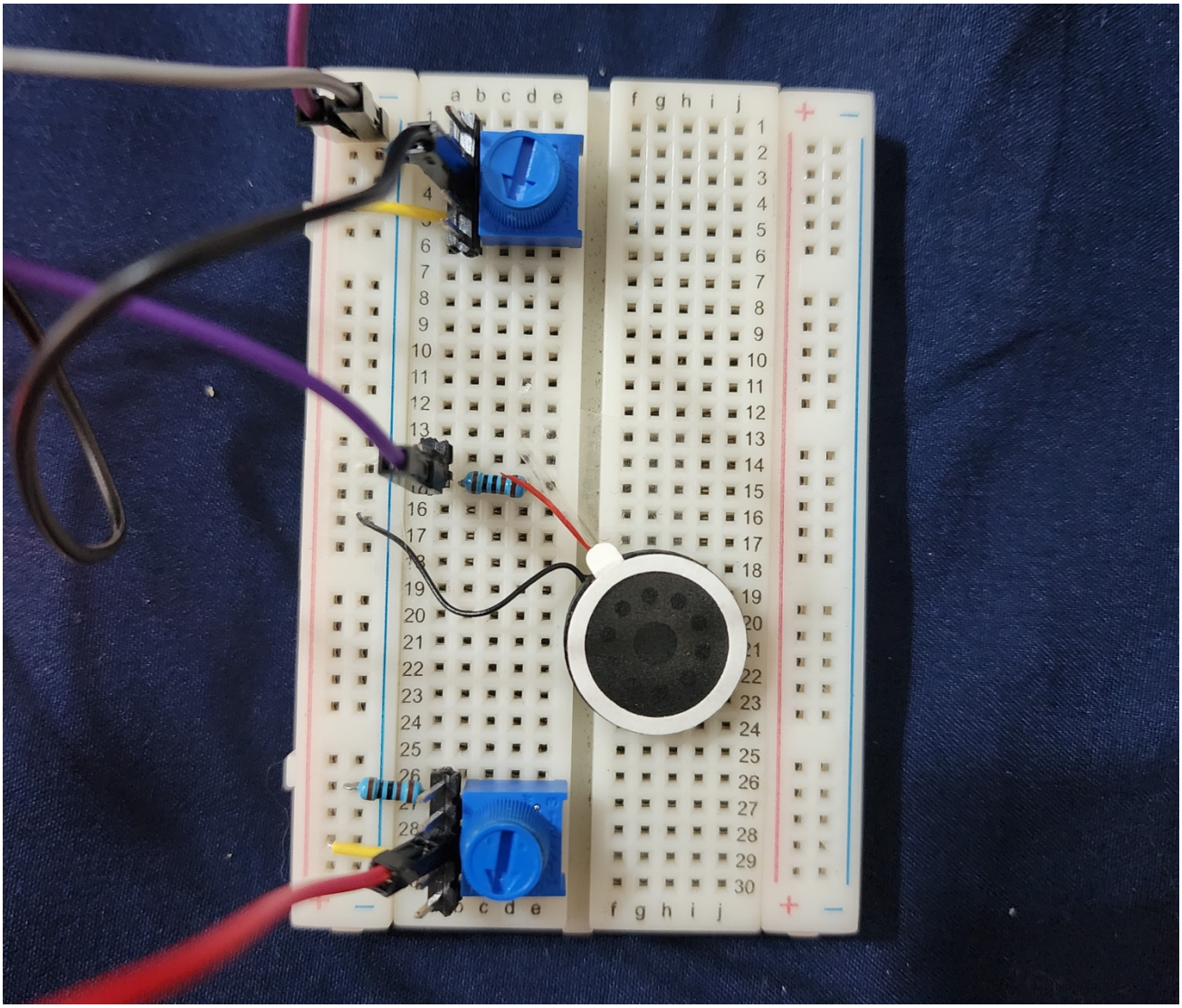

*Figure 4 : Breadboard connection Lab Exercise 2_9*

## Wiring Diagram :

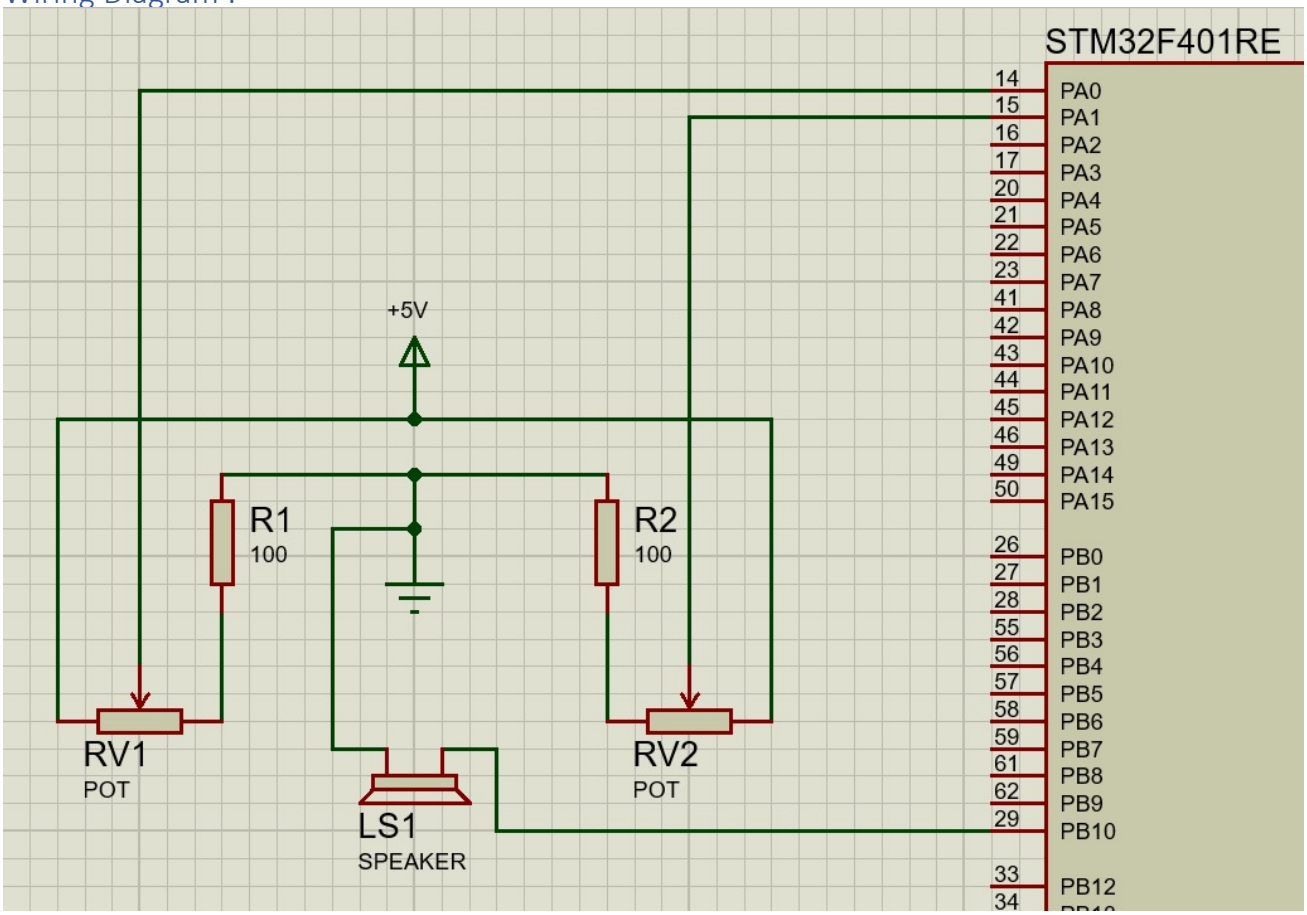


*Figure 5 : Proteus 8 based Wiring Diagram Exercise 2_9*

## Code :

```
/*-----------------------------------------------------------------------------
LAB EXERCISE 9 - Analog input and PWM
 ----------------------------------------
       Use two potentiometers to adjust the volume and pitch of the output sound wave.

       Inputs: Virtual potentiometers 1 and 2
       Output: Virtual speaker,  Real PC

       GOOD LUCK!
 *----------------------------------------------------------------------------*/
/******************************************************************************
* Copyright (C) 2023 by Kiran Jojare & Viraj Patel
*
* Redistribution, modification, or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Kiran Jojare & Viraj Patel and the University of Colorado are not
* liable for any misuse of this material.
*
******************************************************************************/

/**
* @file_name    main.cpp
* @introduction This file demonstrates the use of the mbed API for handling
*               analog input and PWM in an ARM-based microcontroller, focusing on
*               adjusting the volume and pitch of the output sound wave using potentiometers.
*
* @author       Kiran Jojare & Viraj Patel
* @date         Oct 1 2023
* @version      1.0
```

```
*/
#include "mbed.h"
#include "pindef.h"

/*
Define the PWM speaker output
Define analog inputs
Define serial output
*/

//Write your code here
// Define PWM speaker output, analog inputs, and serial output
PwmOut speaker(PB_10);               // PWM Speaker Output
AnalogIn potVolume(PA_0);     // Analog Input for Volume
AnalogIn potPitch(PA_1);      // Analog Input for Pitch
Serial pc(USBTX, USBRX);      // Serial Output to PC

//Define variables
float i;

/*-----------------------------------------------------------------------------
 MAIN function
 *----------------------------------------------------------------------------*/
/**
 * @brief Main Function
 * @details The main function runs an infinite loop where it reads the values from potentiometers,
 *          adjusts the volume and pitch of the speaker output accordingly, and prints the values to
 *          the PC serial terminal. It generates a saw-tooth sound wave with adjustable period and
volume.
 */
int main(){
        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
                */
                // Read the values from the potentiometers
                float volume = potVolume.read();  // Read Volume level
                float pitch = potPitch.read();    // Read Pitch level

                // Calculate frequency based on pitch
                float frequency = 320 + pitch * (8000 - 320);  // Frequency Range: 320Hz to 8000Hz

                // Set speaker period based on calculated frequency
                speaker.period(1.0f / frequency);  // Set the period of the PWM signal

                // Generate a saw-tooth wave and adjust volume
                for (float i = 0; i < 1; i += 0.05f) {
                                speaker.write(i * volume);
                }

                // Print the values to the PC serial terminal
                pc.printf("Volume: %f, Pitch: %f, Frequency: %f Hz\r\n", volume, pitch, frequency);
 }
}
// ********************************ARM University Program Copyright © ARM Ltd
2014*************************************
```

## Lab Questions :

### Question 1:

In our endeavor, we discovered that modifying the increment or decrement value of the variable i is pivotal in defining the properties of the generated waveform. When we augmented this value, we obtained a waveform with diminished points, leading to a probable rough and jagged representation, which may translate to a coarser auditory experience. In contrast, reducing this value results in a waveform abundant in points, yielding it smoother and possibly offering a more nuanced sound

representation. We acknowledged that alterations in the waveform's texture and refinement significantly influence the acoustic perception.

Further, we concluded that discerning these modulations in sound, particularly the pitch, is intricately linked to the alterations in the waveform and the responsiveness of our auditory faculties. In the absence of precise instruments like an oscillator, our reliance was predominantly on our auditory discernment to identify the nuanced changes in the waveform and their auditory repercussions. We concluded that our capabilities were not sufficient to distinguish changes in the waveform just by hearing. This realization emphasized the intricate relationship between the resolution of the waveform and the human ability to perceive subtleties in sound, illustrating the nuanced equilibrium between acoustic representation and perception in sound generation.

# Exercise 2_11 : Serial communication

## Breadboard Connections (Integrated) :

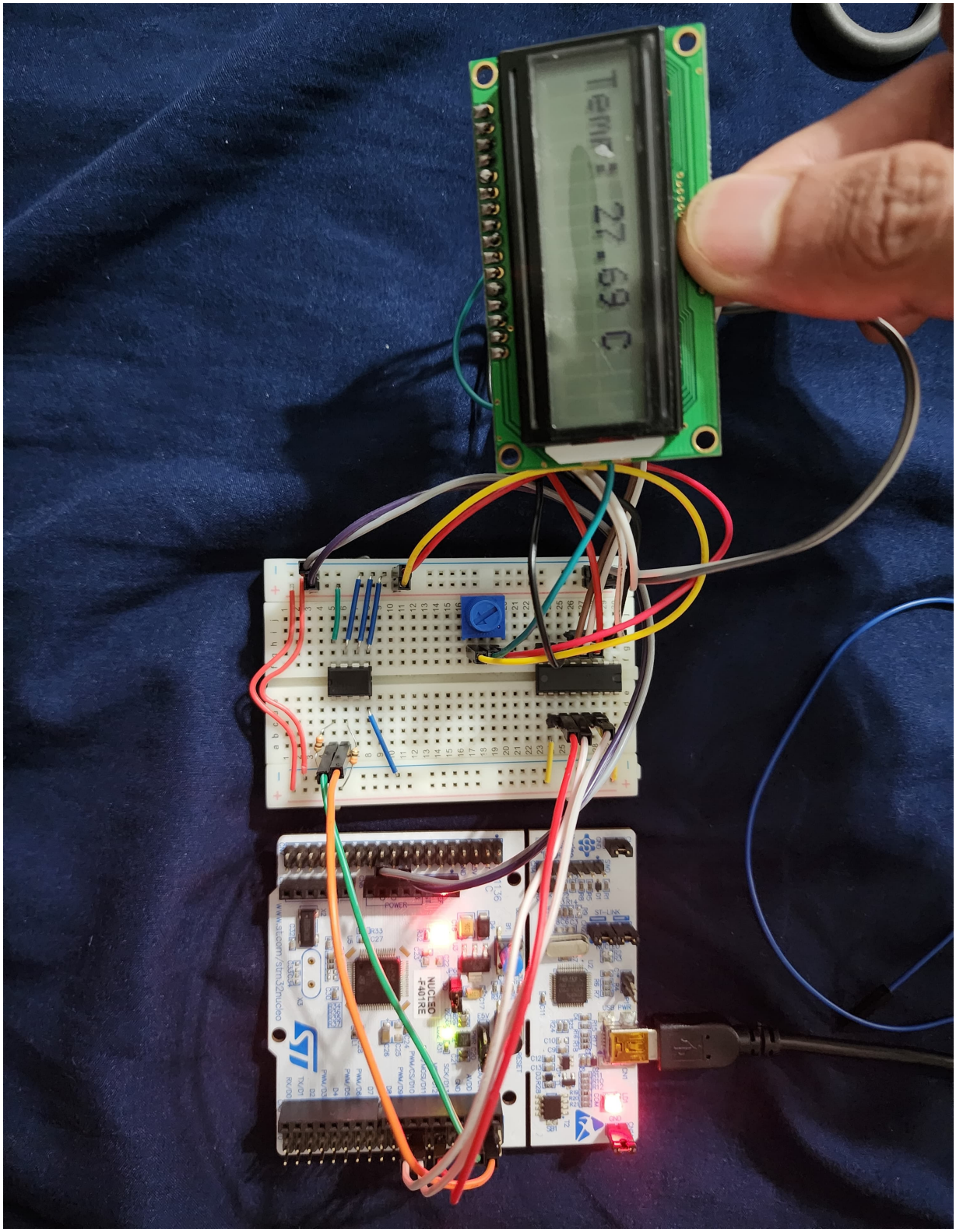


*Figure 6 : Breadboard connection Lab Exercise 2_11 Integrated*

## Wiring Diagram (Integrated):

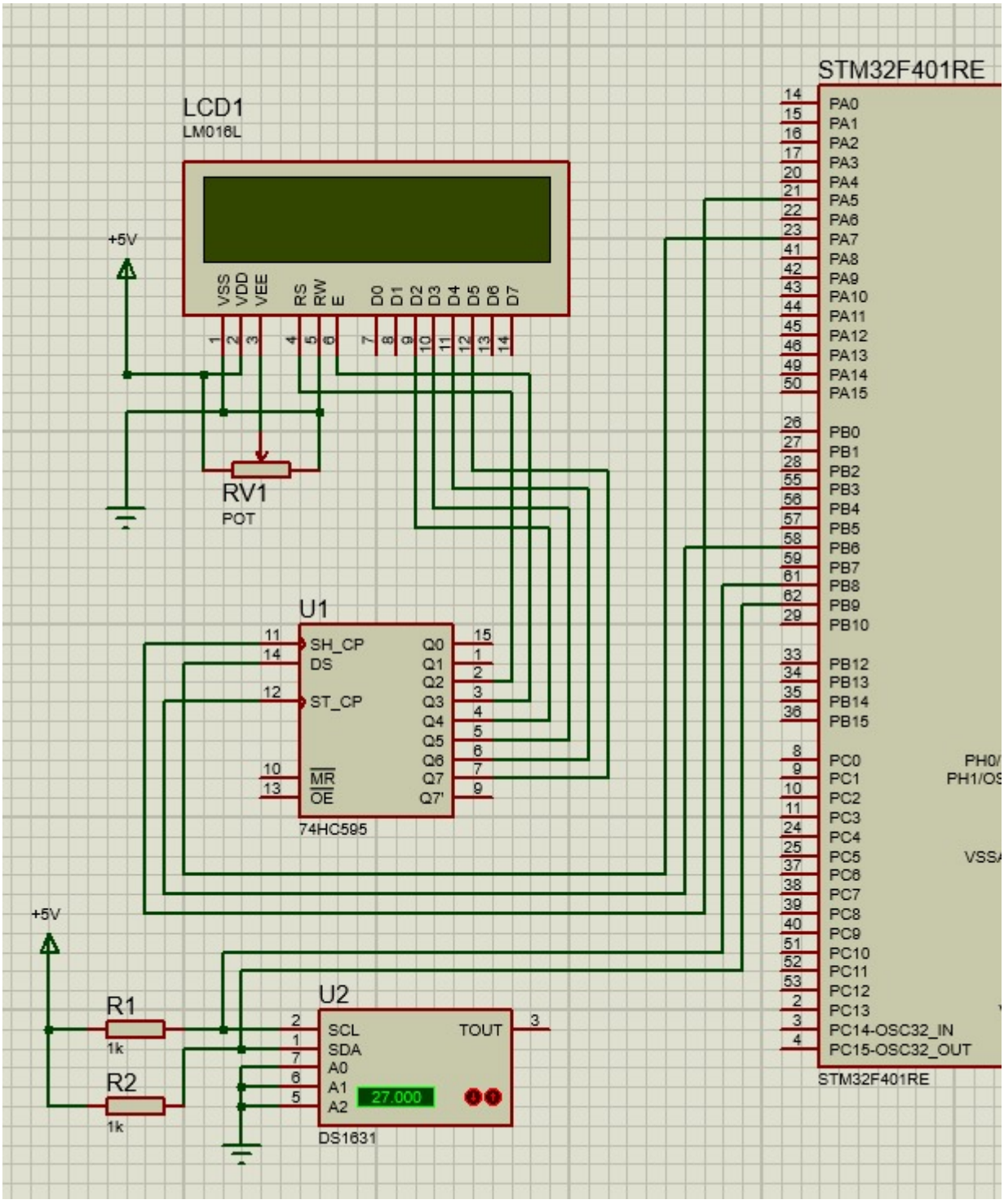


*Figure 7 :: Proteus 8 based Wiring Diagram Exercise 2_11 Integrated*

## Code (Integrated) :

```
/*-----------------------------------------------------------------------------
LAB EXERCISE 11.4- SPI and I2C interface
SERIAL COMMUNICATION
 ---------------------------------------
 Display the temperature from the virtual DS1631 temperature sensor on the
      virtual LCD

 Input: virtual DS1631 temperature sensor
 Output: virtual LCD display

       GOOD LUCK!
 *-----------------------------------------------------------------------------*/

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

//Define the LCD and the temperature sensor
//Write your code here
NHD_0216HZ lcd(SPI_CS, SPI_MOSI, SPI_SCLK);  // Initialize the LCD display.
DS1631 temp_sensor(I2C_SDA, I2C_SCL, 0x90);  // Initialize the temperature sensor with I2C address
0x90.
Serial pc(UART_TX, UART_RX);                 // Initialize UART

//Define a variable to store temperature measurement
float temp;
/*-----------------------------------------------------------------------------
 MAIN function
 *-----------------------------------------------------------------------------*/
int main() {
       //Initialise the LCD

       //Write your code here
       lcd.init_lcd();

       while(1){
              /*
              Read the temperature from the DS1631
              Update the LCD with new temperature measurement
              */

              //Write your code here
              // Read the temperature from the DS1631 sensor.
              temp = temp_sensor.read();

              // Set the baud rate for serial communication
              pc.baud(9600);

              // Clear the LCD screen.
              lcd.clr_lcd();

              // Set the cursor to the beginning of the first line of the LCD.
              lcd.set_cursor(0,0);

              // Print the temperature reading on the LCD screen.
              lcd.printf("Temp: %.2f C", temp);

              // Printing Temperature on UART as well
              pc.printf("Temperature: %.2f Celsius\r\n", temp);

              // Delay to avoid overwhelming the LCD screen with updates.
              wait_ms(1000);

       }
}

// ********************************ARM University Program Copyright (c) ARM Ltd
2014*************************************
```

## Exercise 1 : SPI

In this exercise, we initialised LCD with using the initialisation sequence from the ST7066U LCD driver datasheet) and printed something on LCD.

### Breadboard Connection :

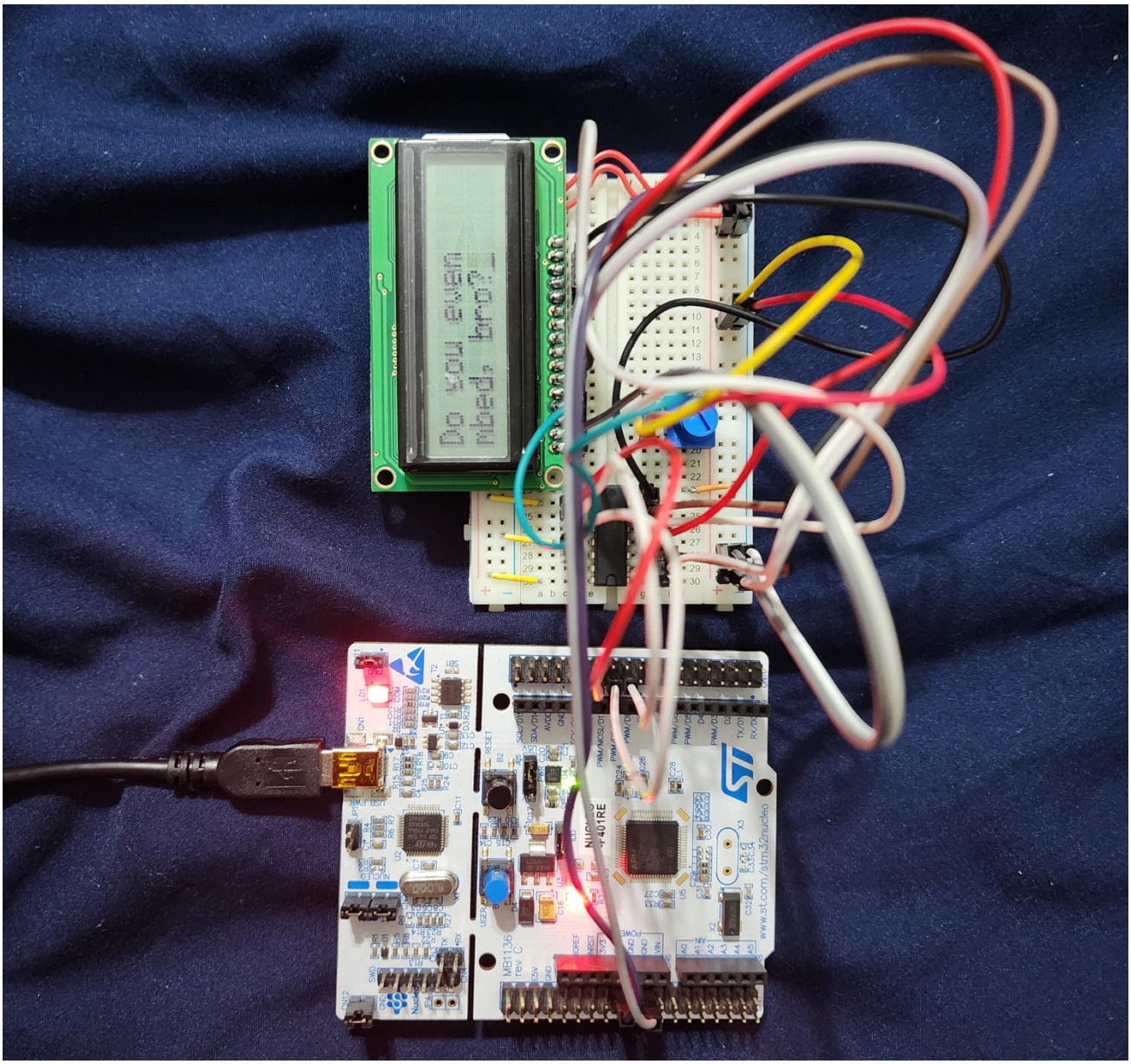


*Figure 8 : Breadboard connection for Exercise 1 SPI for Exercise 2_11*

## Wiring Diagram :

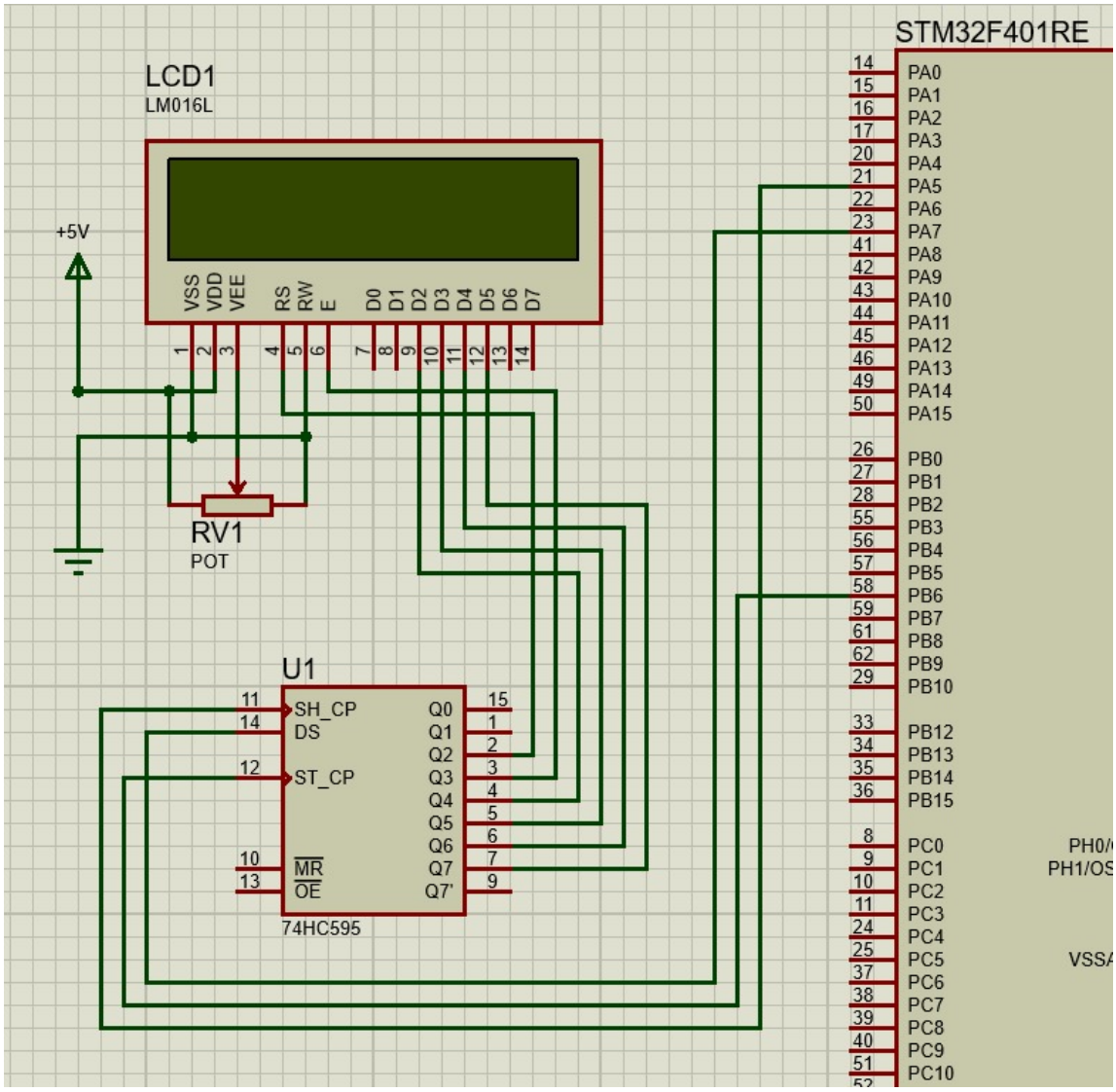


*Figure 9 : Wiring diagram for Exercise 1 SPI for Exercise 2_11*

## Code (SPI) :

### *Main.cpp :*

```
/*-----------------------------------------------------------------------------
LAB EXERCISE 11.1 - SPI interface
SERIAL COMMUNICATION
 ---------------------------------------
 Interface the LCD display using SPI protocol: display four checks (size 10x10
 pixels) at four corners of the LCD.

 Input: None
 Output: LCD display

       GOOD LUCK!
 *----------------------------------------------------------------------------*/
/******************************************************************************
* Copyright (C) 2023 by Kiran Jojare & Viraj Patel
*
* This code snippet is provided with permission for redistribution, modification,
* or use in source or binary forms, with the condition that all the files maintain
* this copyright notice. This is primarily intended for educational purposes,
* allowing users to learn about embedded software development.
* Kiran Jojare, Viraj Patel, and the University of Colorado disclaim liability
```

```
* for any misuse of this material.
*
******************************************************************************/

/**
* @file_name    main.cpp
* @introduction This file serves as the main entry point for the program and contains
*               the logic to initialize SPI and LCD and print messages on the LCD.
*
* @author       Kiran Jojare & Viraj Patel
* @date         Oct 1 2023
* @version      1.0
*/
#include "NHD_0216HZ.h"

/*-----------------------------------------------------------------------------
 MAIN function
 *-----------------------------------------------------------------------------*/

int main() {
        // Initializes the SPI interface.
  init_spi();

  // Initializes the LCD display.
  init_lcd();

  // Printing the string "Do you even" to the LCD.
  print_lcd("Do you even");

  // Moving the cursor to the second line of the LCD.
        set_cursor(0,1);

  // Printing another string "mbed, bro?" to the LCD.
        print_lcd("mbed, bro?");
}

// ********************************ARM University Program Copyright (c) ARM Ltd
2014*************************************
```

## *NHD_0216HZ.c*

```
/*-----------------------------------------------------------------------------
 Newhaven NHD0216HZ LCD C/C++ file
 *-----------------------------------------------------------------------------*/

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

DigitalOut SS(SPI_CS);     //slave select a.k.a. cs or latch for shift reg
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 45 milliseconds: Ensures that the LCD has finished its internal initialization.
    wait_ms(45);

    // 0x30 - Function Set: This is part of the wake-up command and initializing the LCD in 8-bit
mode
    write_cmd(0x30);
    wait_us(40); // Wait for 40 microseconds before sending the next command.

    // 0x20 - Function Set: This command sets the LCD to use a 4-bit data length for subsequent
commands.
    write_cmd(0x20);
```

```
    wait_us(40); // Wait for 40 microseconds before sending the next command.

    // 0x20 - Function Set: Repeating the command as per the initialization sequence.
    write_cmd(0x20);
    wait_us(40); // Wait for 40 microseconds before sending the next command.

    // 0x0C - Display ON/OFF Control: Turns on the display with the cursor off and no blinking.
    write_cmd(0x0C);
    wait_us(40); // Wait for 40 microseconds before sending the next command.

    // 0x01 - Clear Display: Clears the display.
    write_cmd(0x01);
    wait_us(1525); // Longer delay, as clearing the display takes more time.

    // 0x06 - Entry Mode Set: Sets the cursor move direction to increment, and no display shift.
    write_cmd(0x06);
    wait_us(40); // Wait for 40 microseconds before sending the next command.

    // 0x28 - Function Set: Configures the LCD in 4-bit mode, 2 lines, and 5x8 dots format font.
    write_cmd(0x28);
    wait_ms(40); // Wait for 40 milliseconds before sending the next command.

    // Set the cursor to the initial position (0,0).
    set_cursor(0, 0);
}
//Write 4bits to the LCD
void write_4bit(int nibble, int mode) {
    SS = 0;
    spi.write(nibble | ENABLE | mode);
    SS = 1;
    wait_us(1);
    SS = 0;
    spi.write(nibble & ~ENABLE);
    SS = 1;
}
//Write a command to the LCD
void write_cmd(int data) {
    int hi_n;
    int lo_n;

    hi_n = hi_n = (data & 0xF0);
    lo_n = ((data << 4) &0xF0);

    write_4bit(hi_n, COMMAND_MODE);
    write_4bit(lo_n, COMMAND_MODE);
}
//Write data to the LCD
void write_data(char c) {
    int hi_n;
    int lo_n;

    hi_n = hi_n = (c & 0xF0);
    lo_n = ((c << 4) &0xF0);

    write_4bit(hi_n, DATA_MODE);
    write_4bit(lo_n, DATA_MODE);
}
//Set cursor position
void set_cursor(int column, int row) {
    int addr;

    addr = (row * LINE_LENGTH) + column;
    addr |= TOT_LENGTH;
    write_cmd(addr);
}
//Print strings to the LCD
void print_lcd(const char *string) {
    while(*string){
        write_data(*string++);
    }
}

// ******************************ARM University Program Copyright (c) ARM Ltd
2014*************************************
```

## Exercise 2 : Only UART

In this lab exercise we printed mbed only over UART. This implementation doesn't require to implement anything on breadboard hence there is not wiring diagram for this specific task. However I have attached code and serial terminal output as seen below.

### Serial Terminal Output :

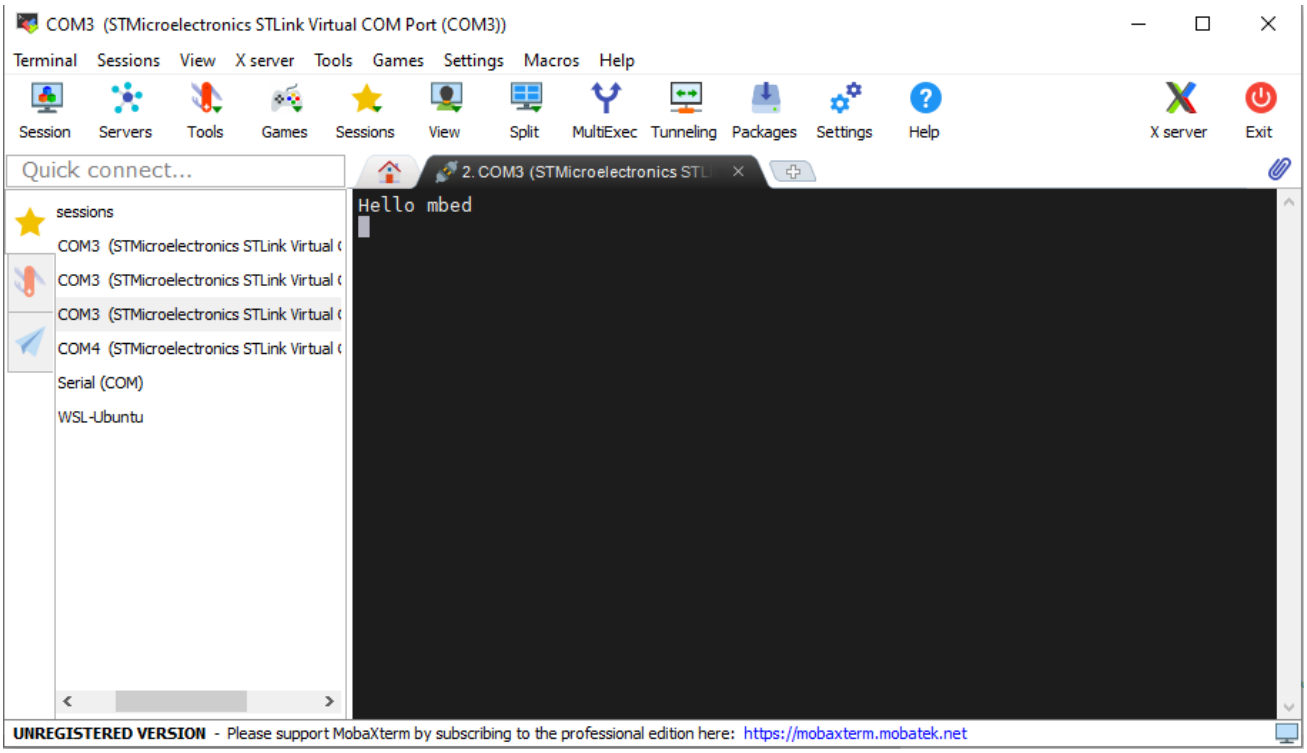


*Figure 10 : Mobaterm Output for Exercise 2 (Only UART)*

### Code(Only UART) :

```
/*-----------------------------------------------------------------------------
LAB EXERCISE 11.2 - UART interface
SERIAL COMMUNICATION
 ---------------------------------------
        Print text to the PC via UART protocol

        Input: None
        Output: PC

        GOOD LUCK!
 *-----------------------------------------------------------------------------*/
/*****************************************************************************
* Copyright (C) 2023 by Kiran Jojare & Viraj Patel
*
* Redistribution, modification, or use of this software in source or binary
* forms is permitted as long as the files maintain this copyright. Users are
* permitted to modify this and use it to learn about the field of embedded
* software. Kiran Jojare & Viraj Patel and the University of Colorado are not
* liable for any misuse of this material.
*
*****************************************************************************/
```

```
/**
* @file_name    main.cpp
* @introduction Describes the functionality of the software, which is to print
*               text to the PC via UART protocol for learning purposes.
*
* @author       Kiran Jojare & Viraj Patel
* @date         Oct 1 2023
* @version      1.0
*/
#include "mbed.h"
#include "pindef.h"

// Serial tx, rx connected to the PC via an USB cable
Serial device(UART_TX, UART_RX);

/*-----------------------------------------------------------------------------
 MAIN function
 *----------------------------------------------------------------------------*/

int main(){
  /*
  Set the baudrate to 9600 bps
  Print "Hello mbed" to the PC serial monitor
  */

  //Write your code here
        // Configuring the Serial object with a baud rate of 9600 bps.
        device.baud(9600);

  // Sending the string "Hello mbed"
  device.printf("Hello mbed\r\n");

}

// ******************************ARM University Program Copyright (c) ARM Ltd
2014************************************
```

## Exercise 3 : I2C UART for Temperature Sensor

### Breadboard Connection :

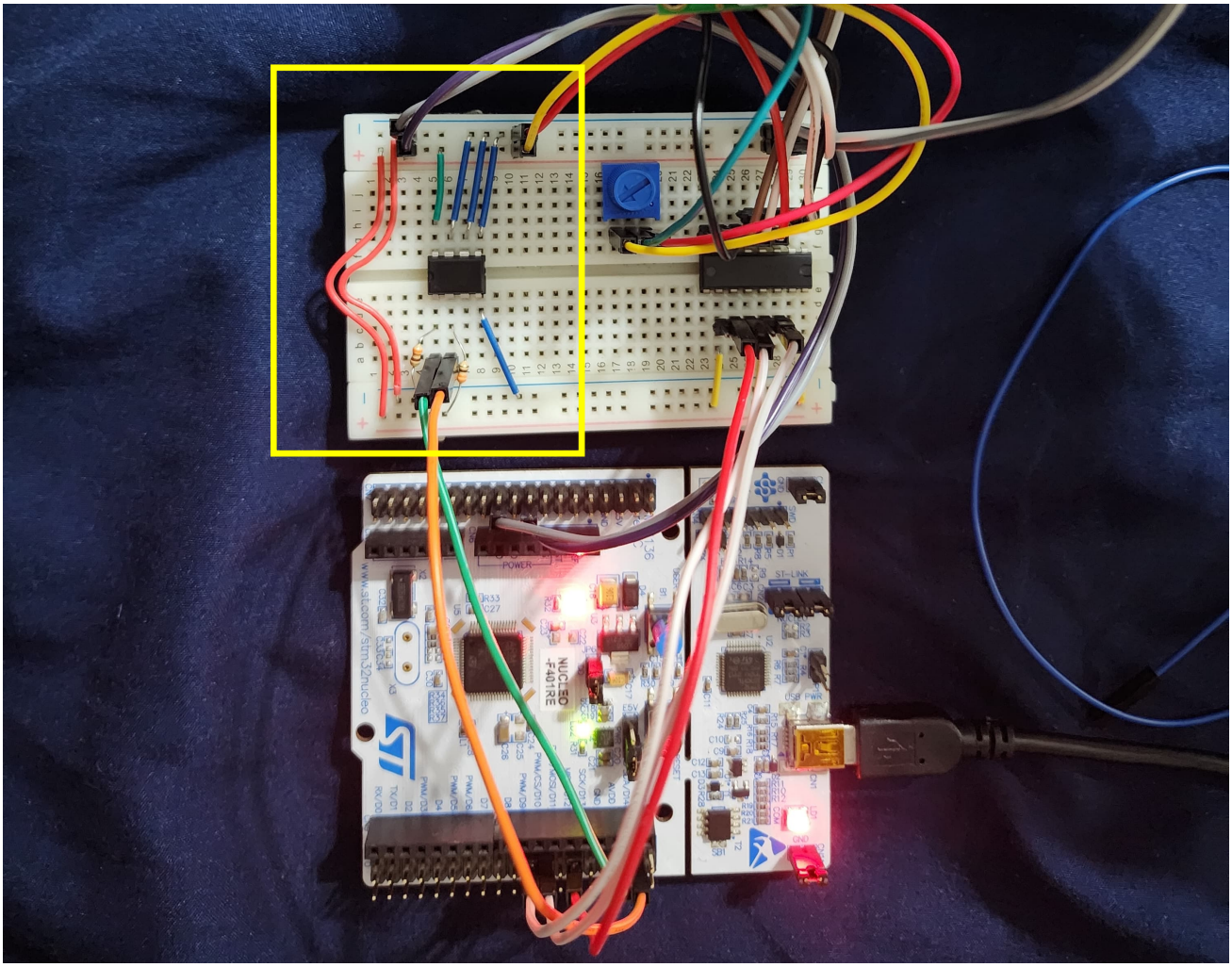

*Figure 11 : Breadboard Connection diagram for Exercise 3 I2C UART Temperature Sensor for Exercise 2_11*

## Wiring Diagram :

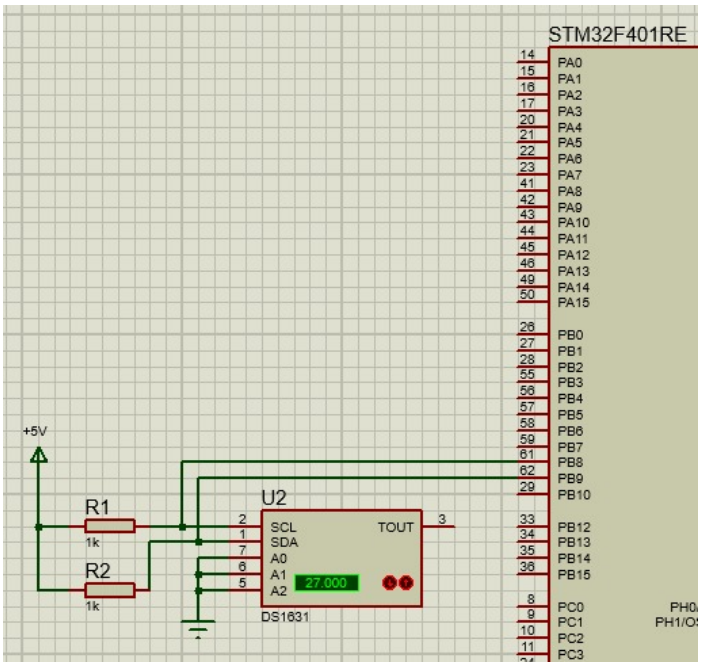


*Figure 12 : Wiring diagram for Exercise 3 I2C UART Temperature Sensor for Exercise 2_11*

## Serial Terminal :

In our experimentation, we observed the output on the serial terminal as we attempted to alter the temperature readings. By placing a warm finger on the I2C IC, we induced a change in temperature, which was subsequently reflected in the serial terminal readings. Before this intervention, the sensor recorded the ambient room temperature as approximately 27 degrees Celsius. This activity helped us observe real-time variations in sensor readings due to external influences, enabling a practical understanding of sensor behaviour and response times.

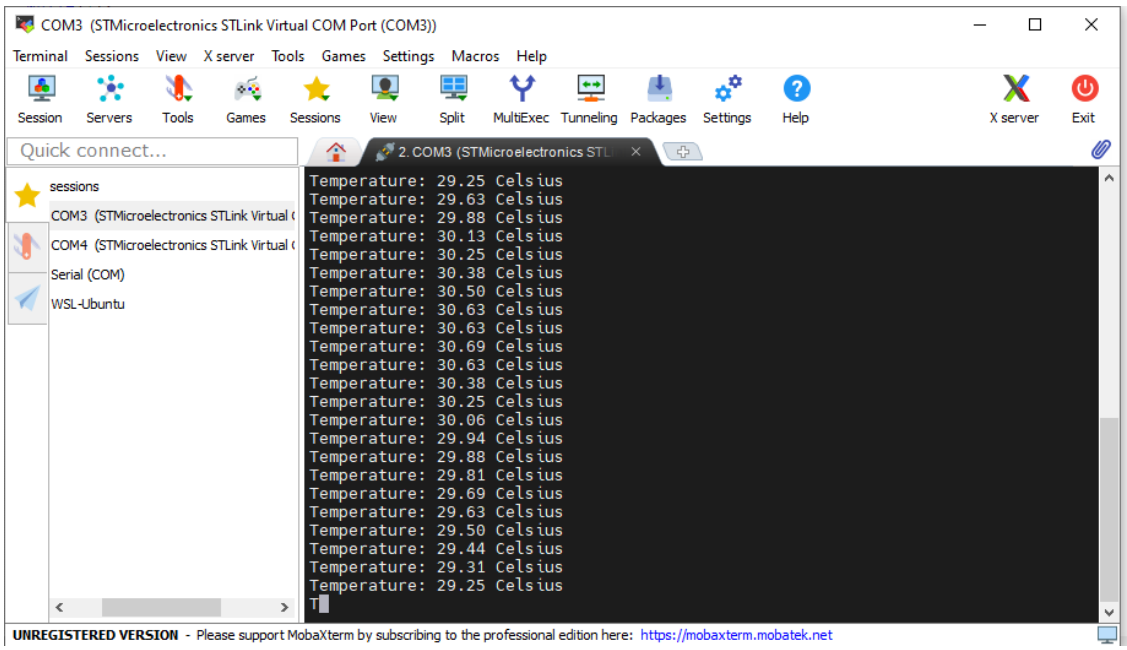


*Figure 13 : Mobaterm Output for Exercise 3 (I2C UART)*

## Code(I2C UART Temperature Sensor) :

```
/*----------------------------------------------------------------------
LAB EXERCISE 11.3 - I2C interface
SERIAL COMMUNICATION
 ---------------------------------------
       Access the virtual temperature sensor via I2C interface, print the current temperature
       to the PC via UART

       Input: temperature sensor
       Output: PC

       GOOD LUCK!
 *---------------------------------------------------------------------------*/
 /*****************************************************************************
* Copyright (C) 2023 by Kiran Jojare & Viraj Patel
*
* This code snippet is provided with permission for redistribution, modification,
* or use in source or binary forms, with the condition that all the files maintain
* this copyright notice. This is primarily intended for educational purposes,
* allowing users to learn about embedded software development.
* Kiran Jojare, Viraj Patel, and the University of Colorado disclaim liability
* for any misuse of this material.
*
*****************************************************************************/

/**
* @file_name    main.cpp
* @description  This file contains the main code for integration of Project 2 Module 2
*
* @author       Kiran Jojare & Viraj Patel
* @date         Oct 1 2023
* @version      1.0
*/
#include "mbed.h"
#include "pindef.h"

//I2C interface
I2C temp_sensor(I2C_SDA, I2C_SCL);

// UART interface
Serial pc(UART_TX, UART_RX);

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

// Array storing the Start Convert T command and the Read Temperature command.
char cmd[] = {0x51, 0xAA};

// Array to store the 16-bit temperature data read from the sensor.
char read_temp[2];

/*----------------------------------------------------------------------
 MAIN function
 *---------------------------------------------------------------------------*/

int main(){
       // Set the baud rate for serial communication
       pc.baud(9600);

       while(1){
              /*
              Write the Start Convert T command to the sensor
              Write the Read Temperature command to the sensor
              Read the 16-bit temperature data
              */

              //Write your code here
              // Send the Start Convert T command to the sensor.
              temp_sensor.write(temp_addr, &cmd[0], 1);

              // Short delay to give sensor time to process the command and start conversion
              wait_ms(10);

              // Send the Read Temperature command to the sensor.
```

```
            temp_sensor.write(temp_addr, &cmd[1], 1);

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

            //Convert temperature to Celsius
            float temp = (float((read_temp[0] << 8) | read_temp[1]) / 256);

            //Print temperature to the serial monitor

            //Write your code here
            // Print the calculated temperature to the serial monitor.
            pc.printf("Temperature: %.2f Celsius\r\n", temp);

            // Delay to avoid overwhelming output.
            wait_ms(1000);

        }
}

// ********************************ARM University Program Copyright (c) ARM Ltd
2014*************************************
```

## Lab Questions :

### Question 1: What temperature is displayed on your PC terminal window? What is displayed on the LCD ?

The temperature displayed on PC terminal window was exactly **27 Celsius** as seen in figure 12. Same **27 Celsius** was also displayed on LCD as seen in figure 6 and figure 8.