

Exercise: RTOS THREADS

Breadboard Connections :

Please locate the breadboard connections required for implementing the Exercise specifications. The figure includes added LED and Potentiometer components as highlighted in yellow, while all other elements remain consistent with the previous assignment.

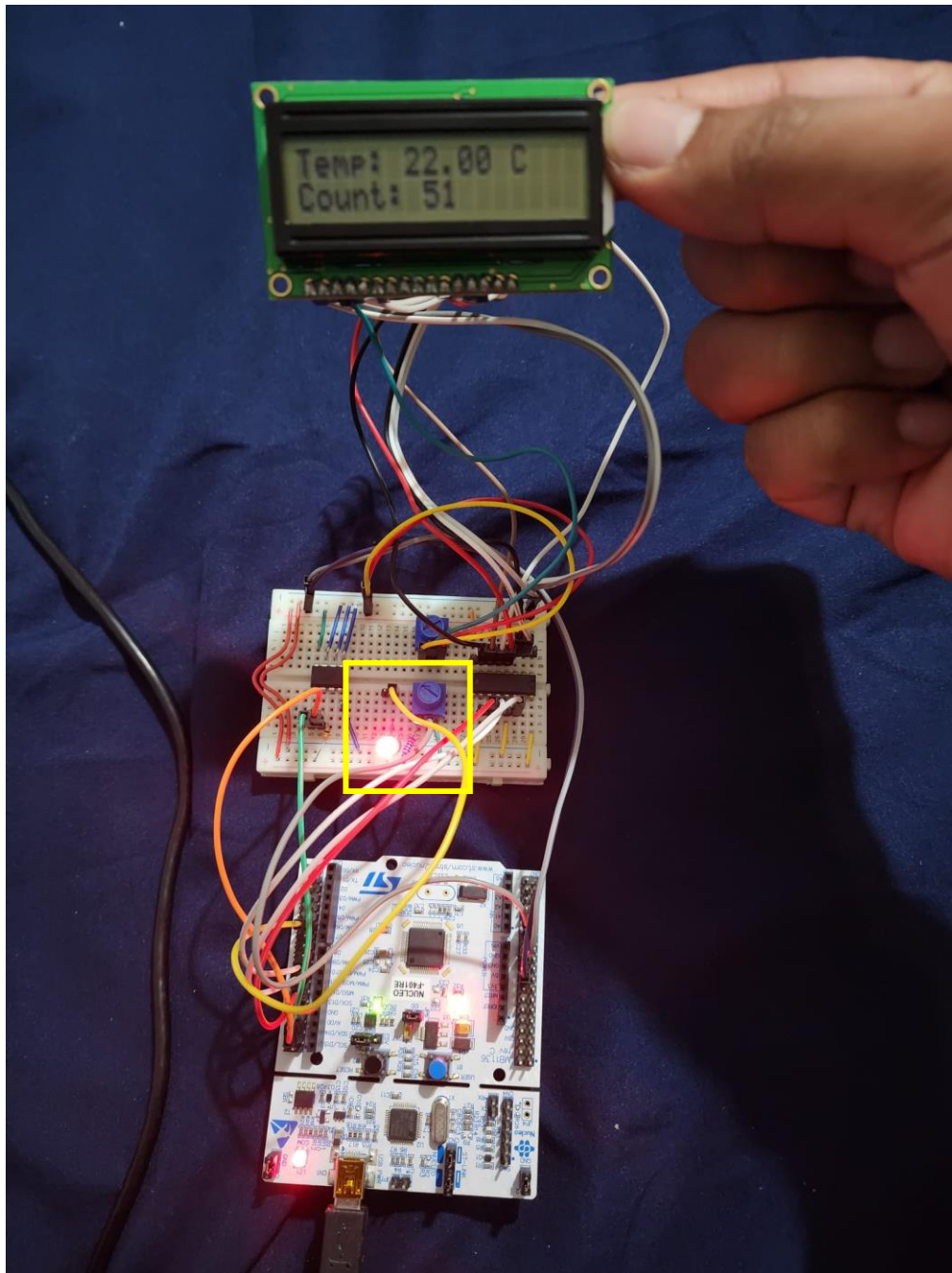


Figure 1: Breadboard connection Lab for Project 1 Module 3

Wiring Diagram:

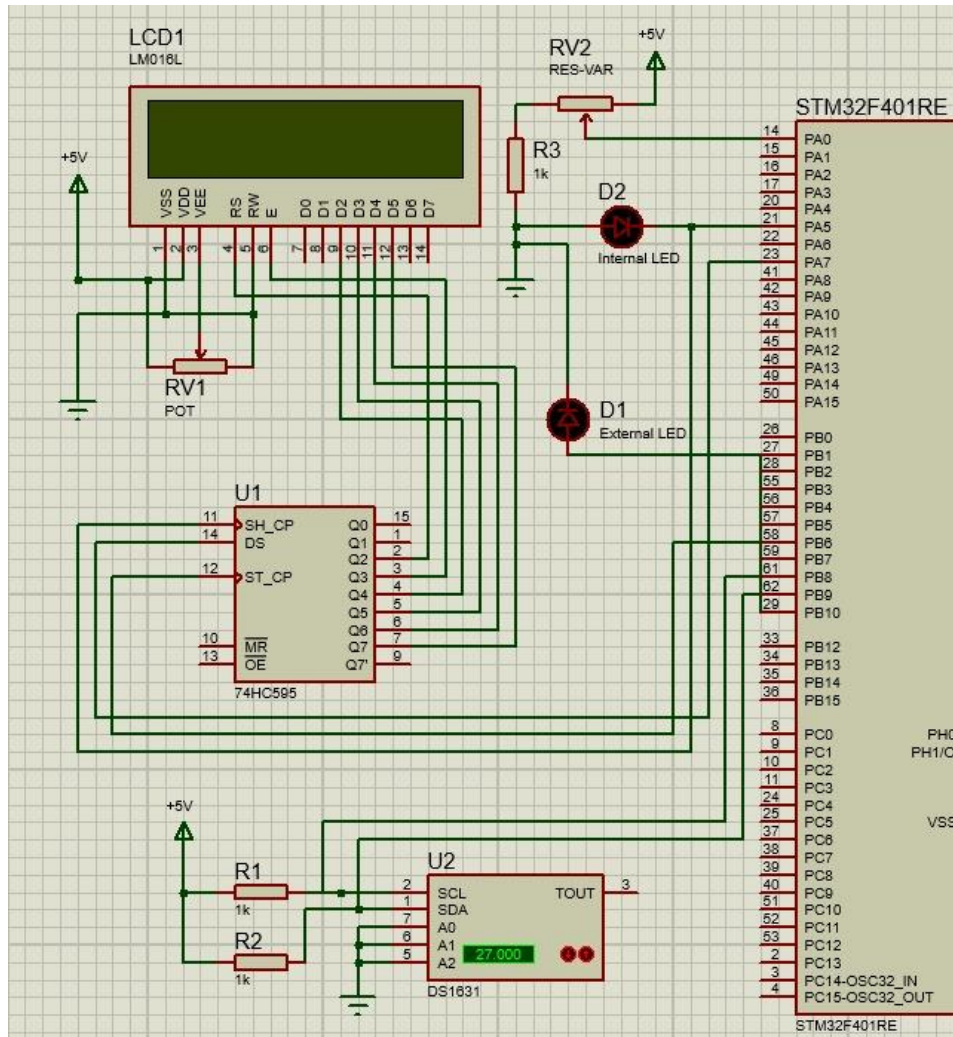


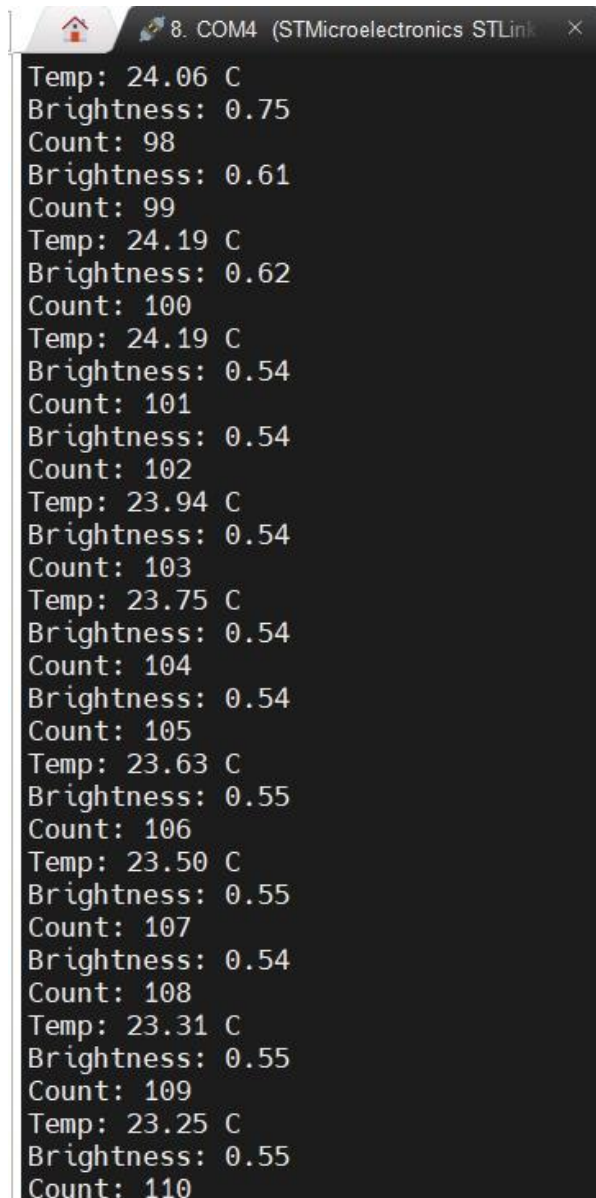
Figure 2 : Proteus 8-based Wiring Diagram for Project 1 Module 3

- Note:

The pin PA5 serves a dual purpose, functioning as both the internal LED PA2 and as an input to the shift register. We couldn't employ any alternative LED because all LEDs from the mbed API were internally linked to PA5. The thread responsible for blinking an LED utilized the internal LED, while the external LED was employed to regulate brightness. Ideally, we would have preferred to use both LEDs as external ones, but we only had a single external LED available.

Serial Terminal

The serial terminal screenshot provided illustrates the concurrent execution of tasks, namely displaying a count on an LCD screen, showing the temperature on the same screen, and simultaneously adjusting the screen's brightness using a potentiometer through a multithreaded approach.



```

8. COM4 (STMicroelectronics STLink)
Temp: 24.06 C
Brightness: 0.75
Count: 98
Brightness: 0.61
Count: 99
Temp: 24.19 C
Brightness: 0.62
Count: 100
Temp: 24.19 C
Brightness: 0.54
Count: 101
Brightness: 0.54
Count: 102
Temp: 23.94 C
Brightness: 0.54
Count: 103
Temp: 23.75 C
Brightness: 0.54
Count: 104
Brightness: 0.54
Count: 105
Temp: 23.63 C
Brightness: 0.55
Count: 106
Temp: 23.50 C
Brightness: 0.55
Count: 107
Brightness: 0.54
Count: 108
Temp: 23.31 C
Brightness: 0.55
Count: 109
Temp: 23.25 C
Brightness: 0.55
Count: 110
  
```

Figure 3 : Serial Terminal Output

Lab Questions:

- What temperature is displayed on the LCD?

As depicted in Figure 1, the breadboard setup displayed a temperature reading of 22 degrees Celsius on the LCD. Additionally, it's important to observe that the temperature indicated in the serial terminal output fluctuates, reflecting our attempts to modify the temperature by exposing the temperature sensor to a warm source.

Code

```
/*-----  
LAB EXERCISE 12 - Real-Time Operating System  
-----  
    Integrate functions developed in previous modules and run then concurrently  
    in the mbed RTOS. The following four threads have to be implemented:  
  
    1. Display the temperature on the LCD and to the PC  
    2. Adjust the brightness of the RGB LED using a potentiometer  
    3. Display an incrementing counter on the LCD and to the PC  
    4. Blink an LED  
  
    NOTE: For this lab, the LCD, temp sensor and POT are virtual devices.  
  
    GOOD LUCK!  
-----*/  
/*****  
* Copyright (C) 2023 by Viraj & Kiran  
*  
* 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. Viraj, Kiran and the University of Colorado Boulder are not  
* liable for any misuse of this material.  
*  
*****/  
  
/**  
* @file main.cpp  
* @brief This file integrates various functions to achieve multi-threading  
* on mbed using its RTOS.  
*  
* Project 1 Module 3  
* LAB EXERCISE 12 - RTOS  
*  
* @author Viraj, Kiran  
* @date Oct 7 2023  
* @version 1.0  
*/  
#include "mbed.h"  
#include "rtos.h"  
#include "DS1631.h"  
#include "NHD_0216HZ.h"  
#include "pindef.h"  
  
#include <stdint.h>  
/*  
Define the mutex  
Define the LCD display and the temperature sensor  
Define other inputs and outputs  
*/  
  
//Write your code here  
// Init LCD, I2C & UART  
  
// LCD setup using SPI communication.  
NHD_0216HZ lcd(SPI_CS, SPI_MOSI, SPI_SCLK);  
  
// DS1631 temperature sensor setup using I2C communication.  
DS1631 temp_sensor(I2C_SDA, I2C_SCL, 0x90);  
  
// Serial communication setup using UART for debugging.  
Serial pc(UART_TX, UART_RX);  
  
// Internal LED setup with default state as OFF.  
DigitalOut ILED(LED2, 0);  
  
// Potentiometer setup to control external LED brightness.  
AnalogIn ELED_Bright(PA_0);  
  
// PWM setup for controlling external LED brightness.
```

```
PwmOut ELED(PB_10);

// Global variables for temperature and counter.
float temp;
int count = 0;

// Mutexes for synchronized access among threads.
Mutex mymutex1, mymutex2, mymutex3, mymutex4;

/**
 * @brief Displays the temperature on the LCD and sends the reading over UART.
 *
 * The thread function, when invoked, reads the temperature sensor value
 * and displays it on the LCD. The temperature reading is also transmitted
 * over UART for debugging or external display purposes.
 */
void temp_thread(void const *args) {
    while (1) {
        // Introduce a delay of 1 second (1000 milliseconds).
        Thread::wait(1000);

        // Acquire a lock on the mutex to ensure exclusive access to shared resources.
        mymutex1.lock();

        // Read the temperature from the sensor.
        temp = temp_sensor.read();

        // Display the current temperature value on the LCD.
        lcd.set_cursor(0,0); // Position the cursor at the beginning of the first line.
        lcd.printf("Temp: %.2f C", temp);

        // Transmit the temperature value over UART for monitoring purposes.
        pc.printf("Temp: %.2f C\r\n", temp);

        // Release the mutex lock, allowing other threads to access shared resources.
        mymutex1.unlock();

        // Introduce a sleep to allow other threads to execute.
        sleep();
    }
}

/**
 * @brief Adjusts the brightness of the External LED using a potentiometer.
 *
 * The thread function, when invoked, reads the potentiometer value
 * and adjusts the external LED's brightness accordingly. The brightness value
 * is also sent over UART for debugging or external display.
 */
void adjust_brightness(void const *args) {
    while (1) {
        // Introduce a delay of 1 second (1000 milliseconds).
        Thread::wait(1000);

        // Acquire a lock on the mutex to ensure exclusive access to shared resources.
        mymutex2.lock();

        // Read the potentiometer value, which will be in the range of 0.0 to 1.0.
        float potValue = ELED_Bright.read();

        // Adjust the external LED's brightness based on the potentiometer value.
        ELED.write(potValue);

        // Send the brightness value over UART for monitoring purposes.
        pc.printf("Brightness: %0.2f\n\r", potValue);

        // Release the mutex lock, allowing other threads to access shared resources.
        mymutex2.unlock();

        // Introduce a sleep to allow other threads to execute.
    }
}
```

```
        sleep();
    }
}

/**
 * @brief Blinks an internal LED with a 1-second interval.
 *
 * The thread function, when invoked, turns the internal LED on and off
 * with a 1-second interval, creating a blinking effect.
 */
void led1_thread(void const *args) {
    while (1) {
        // Acquire a lock on the mutex to ensure exclusive access to shared resources.
        mymutex3.lock();

        // Introduce a delay of 1 second (1000 milliseconds).
        Thread::wait(1000);

        // Turn on the internal LED.
        ILED = 1;

        // Wait for another second.
        Thread::wait(1000);

        // Turn off the internal LED, completing the blink cycle.
        ILED = 0;

        // Release the mutex lock, allowing other threads to access shared resources.
        mymutex3.unlock();

        // Introduce a sleep to allow other threads to execute.
        sleep();
    }
}

/**
 * @brief Displays an incrementing counter on the LCD and via UART.
 *
 * The thread function, when invoked, continuously increments a counter
 * and displays its value both on an LCD screen and through UART communication.
 */
void count_thread(void const *args) {
    while (1) {
        // Introduce a delay of 1 second (1000 milliseconds)
        Thread::wait(1000);

        // Acquire a lock on the mutex to ensure exclusive access to shared resources.
        mymutex4.lock();

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

        // Display the current value of the counter on the LCD.
        lcd.printf("Count: %d", count);

        // Also send the counter's value over UART for debugging or external display.
        pc.printf("Count: %d\n\r", count);

        // Increment the counter value for the next cycle.
        count++;

        // Release the mutex lock, allowing other threads to access shared resources.
        mymutex4.unlock();

        // Introduce a sleep to allow other threads to execute.
        sleep();
    }
}
```



```
// Define the thread for the temperature display with normal priority and the default stack size.
osThreadDef(temp_thread, osPriorityNormal, DEFAULT_STACK_SIZE);
// Define the thread for adjusting LED brightness with normal priority and the default stack size.
osThreadDef(adjust_brightness, osPriorityNormal, DEFAULT_STACK_SIZE);
// Define the thread for blinking the LED with normal priority and the default stack size.
osThreadDef(led1_thread, osPriorityNormal, DEFAULT_STACK_SIZE);
// Define the thread for displaying the counter with normal priority and the default stack size.
osThreadDef(count_thread, osPriorityNormal, DEFAULT_STACK_SIZE);

/*-----
MAIN function
-----*/
/**
 * @brief Main function initializes LCD and starts all threads.
 */
int main() {
    /*
        Initialise and clear the LCD display
        Start all threads
        Wait for timer interrupt
    */

    //write your code here
    // Initialize the LCD
    lcd_init_lcd();    // Initialize the LCD display.
    lcd_clr_lcd();     // Clear the LCD for fresh display.

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

    // NOTE: The following lines are commented out because when we tried using them,
    // we encountered a HardFault due to memory allocation issues. Thus, we resorted to using
    // osThreadCreate() for a more stable performance. But still keeping it in the code just to
    // let you know we tired debugging it. We found out that using osThreadCreate() memory was
    // allocated statically compared to other way resulting in which memory was getting allocated
    // dynamically

    // Thread thread1(temp_thread, NULL, osPriorityNormal, DEFAULT_STACK_SIZE, NULL);
    // Thread thread2(adjust_brightness, NULL, osPriorityNormal, DEFAULT_STACK_SIZE, NULL);
    // Thread thread3(led1_thread, NULL, osPriorityNormal, DEFAULT_STACK_SIZE, NULL);
    // Thread thread4(count_thread, NULL, osPriorityNormal, DEFAULT_STACK_SIZE, NULL);

    // Creation of threads using osThreadCreate().
    osThreadCreate(osThread(temp_thread), NULL);
    osThreadCreate(osThread(adjust_brightness), NULL);
    osThreadCreate(osThread(led1_thread), NULL);
    osThreadCreate(osThread(count_thread), NULL);

    // Infinitely wait for other threads to execute.
    Thread::wait(osWaitForever);

    return 0;
}

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