***Embedded Systems Essentials with Arm:  
Get Practical with Hardware***

**LAB 3**

**Introducing the Real-Time Operating System**

Contents

[1 Introduction 1](#_Toc67644459)

[2 Resources 1](#_Toc67644460)

[3 Configuring the Circuit 2](#_Toc67644461)

[4 Programming with the RTOS 4](#_Toc67644462)

[4.1 Introducing the RTOS 4](#_Toc67644463)

[4.2 A Two-Thread Program 4](#_Toc67644464)

[4.3 Introducing the Temperature Thread, and the Mutex 6](#_Toc67644465)

[4.4 Adding the LED Brightness Thread 7](#_Toc67644466)

[4.5 Troubleshooting 8](#_Toc67644467)

[5 Conclusion 8](#_Toc67644468)

[6 References 8](#_Toc67644469)

# Introduction

This lab develops directly from Lab 2, with very similar hardware. The major step forward is in implementing the Mbed Real Time Operating System (RTOS). The overarching aim (and it’s a big one!) for the lab is to implement basic RTOS concepts, leading to a multi-tasking program running on an RTOS.

We will develop a program which runs four tasks or threads:

1. Read and display temperature on the LCD (Liquid Crystal Display)
2. Display an incrementing counter on the LCD
3. Adjust the brightness of an LED using a potentiometer
4. Blink an LED

# Resources

In this lab, for program development, you can use either the Mbed Studio, or the on-line compiler (or both). Both were introduced in Lab 0 and used in Labs 1 and 2. When programming, remember the on-line and print references which were introduced in Lab 0; they are repeated here as References 1-7.

The hardware elements needed are listed in Table 1. Items which are in grey were used in previous labs, and continue to be needed. Those in black are new for this lab.

|  |  |
| --- | --- |
| **Item** | **Qty.** |
| STM32F401 Nucleo-64 Development Board | 1 |
| Bread Board | 2 |
| Jumper Wires (kit) | 1 |
| LED with internal current-limiting resistor | 3 |
| 74HC595N Shift Register | 1 |
| Newhaven LCD. NHD-0420H1Z-FSW-GBW-33V3 | 1 |
| Maxim DS1631 Temperature sensor | 1 |
| 1 kΩ resistor | 2 |
| 10kΩ potentiometer | 2 |
| 100 Ω resistor | 1 |

*Table 1: List of Required Parts*

# Configuring the Circuit

Connect up the circuit of Figure 1. Hopefully you have kept your build from Lab 2, so it’s simply a question of adding the potentiometer circuit and the LED. The connection details of the Nucleo board are included in Figure 2 for convenience. A completed build is shown in Figure 3. The potentiometer used here plugs directly into the breadboard.

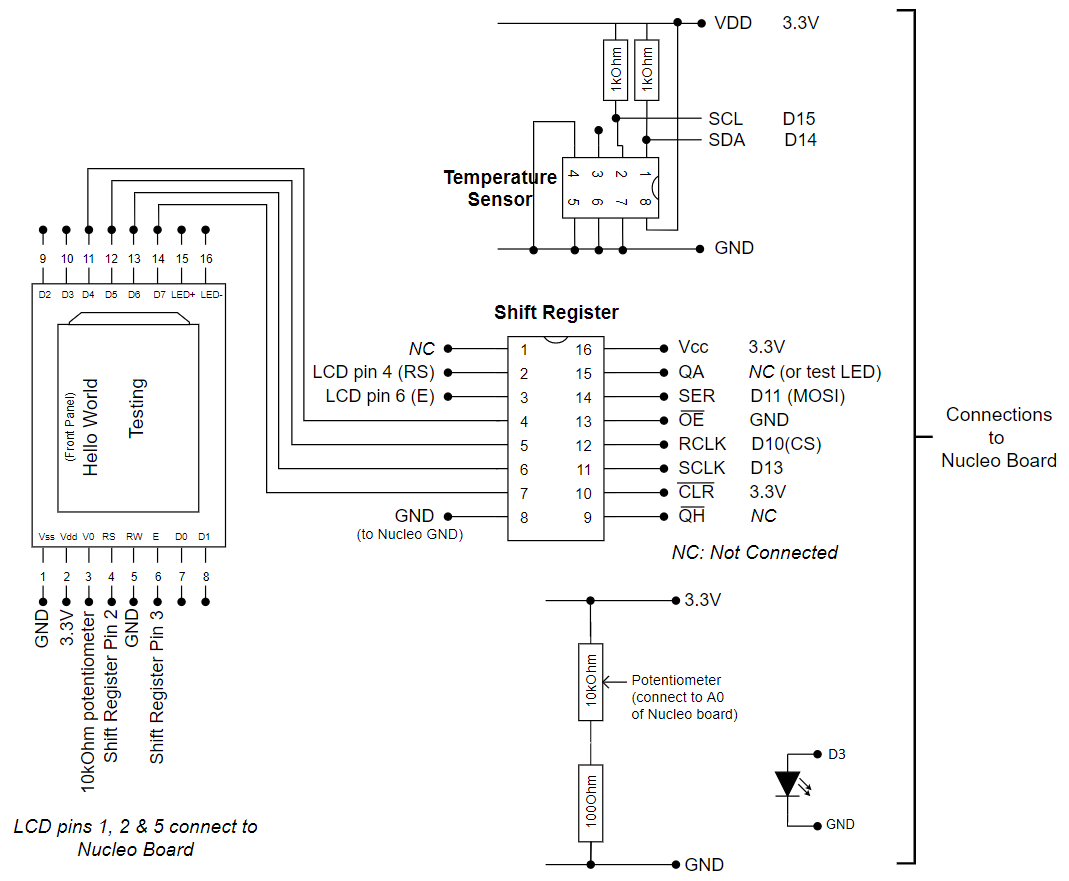


Figure 1: Circuit Layout

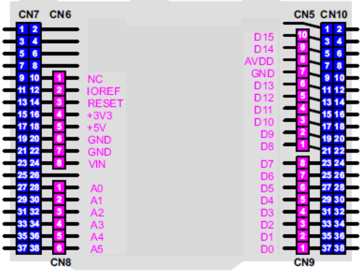
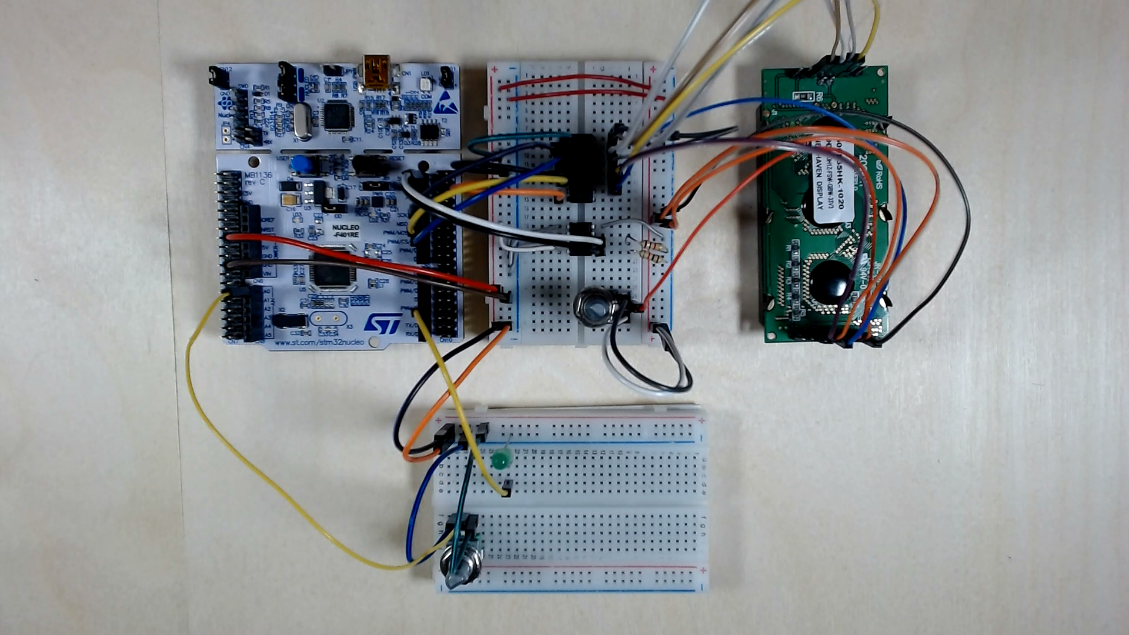


Figure 2: Connections on Nucleo Board, Arduino Connectors Only



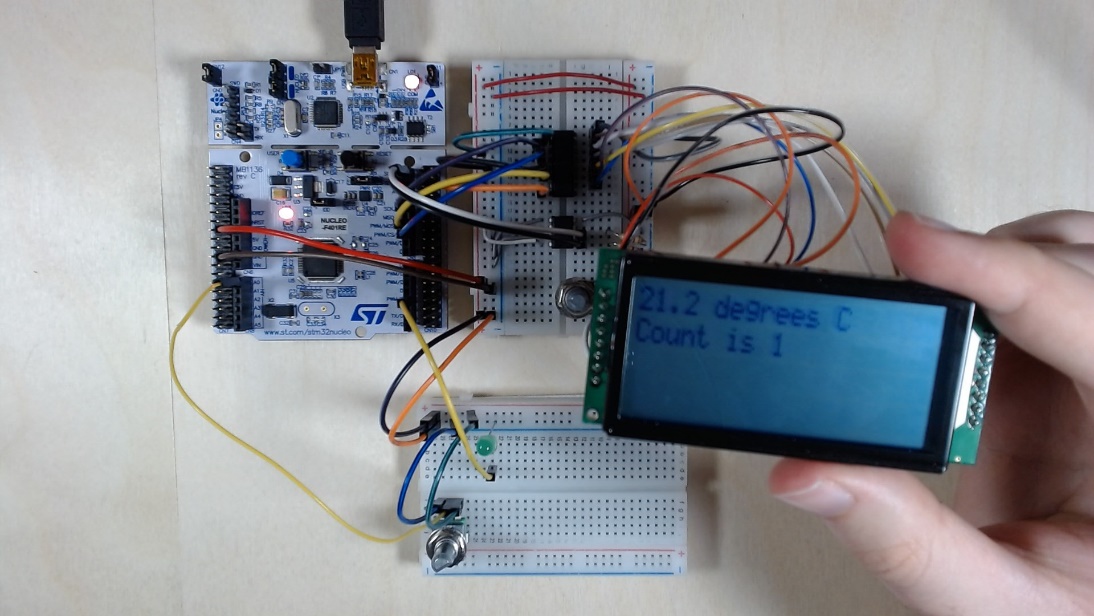


Figure 3: A Completed Circuit Build

# Programming with the RTOS

In this lab, we will progressively integrate tasks or threads into a program, using the Mbed RTOS, up to a maximum of four. The threads will then run concurrently.

## Introducing the RTOS

The Mbed RTOS APIs handle all aspects of real-time programming when used with any Mbed-enabled components. In this lab, we introduce the Thread and Mutex elements of the Mbed RTOS. The Thread class allows defining, creating and controlling parallel tasks. A Mutex is used to synchronize the execution of threads, for example to protect the access to a shared resource.

## A Two-Thread Program

We start with a program having only two tasks or threads - blinking an LED, and writing a counter value to the LCD. The program is shown in Program Example 1. The thought-process in writing with an RTOS is different from conventional sequential programming. Now we write each task or thread as its own function, though it may call other functions. Each thread function should be structured as an endless loop, so program execution can pick up within that loop as the RTOS gives it time.

Although it may initially appear complex, look carefully through the program, and identify all its main features. The two threads appear below the heading “Threads”. It should be possible to understand both of these without too much difficulty. The **main()** function first initialises the LCD, then launches each task with a “callback”. The general format of this is **thread.start(callback(task, argument))**. The program then waits in a permanent loop for interrupts. It is the timer interrupt which initiates the RTOS task switch. Because we’re using the LCD, we find all its associated drive functions below the “LCD Functions” heading, as used in previous labs.

/\* ---------------------------------------------------------------------

Module 3: Temperature Sense Unit with RTOS

Two threads are implemented:

1. Display an incrementing counter on the LCD 2. Blink the on-board LED

\*---------------------------------------------------------------------\*/

#include "mbed.h"

#define ENABLE 0x08 // ORed in to data value to strobe E bit

#define COMMAND\_MODE 0x00 //clears LCD RS line, for command transfer

#define DATA\_MODE 0x04 //sets LCD RS line to 1, for data transfer

DigitalOut led(LED1); //Nucleo LED

DigitalOut CS(D10); //SPI Chip Select

SPI ser\_port(D11, D12, D13); //Initialise SPI, using default settings

// Temp sensor variables

I2C temp\_sensor(I2C\_SDA, I2C\_SCL); //Configure I2C interface

BufferedSerial pc(USBTX, USBRX, 9600);

const int temp\_addr = 0x90; //I2C address of temperature sensor DS1631

char commands[] = {0x51, 0xAA};

char read\_temp[2];

char LCD\_result[9]; //holds result to be displayed on LCD.

//Function Prototypes

void clr\_lcd(void);

void init\_lcd(void);

void print\_lcd(const char \*string);

void shift\_out(int data);

void write\_cmd(int cmd);

void write\_data(char c);

void write\_4bit(int data);

//-------------- Threads -----------------//

Thread thread1; //Blink an LED

void led1\_thread(void const \*args){

led = 0;

while(true) {

led = !led;

thread\_sleep\_for (500);

}

}

Thread thread2;

//Display a counter on the LCD

void count\_thread(void const \*args){

char k = 0;

while(1) {

write\_cmd(0xc0); //set cursor to start of second line

wait\_us(40);

print\_lcd("Count is ");

write\_data(k|0x30); //OR in 0x30 to produce the ASCII code for k

k++;

if (k>9)k=0;

thread\_sleep\_for (1000);

}

}

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

int main(){

init\_lcd(); //Initialise the LCD display

clr\_lcd(); //Clear the LCD

thread1.start(callback(led1\_thread,&led));

thread2.start(callback(count\_thread,&ser\_port));

while(true) { //Wait for timer interrupt

\_\_wfi();

}

}

//----------------- LCD functions---------------------//

void init\_lcd(void){ //follow designated procedure in data sheet

thread\_sleep\_for (40);

shift\_out(0x30); //function set 8-bit

wait\_us(37);

write\_cmd(0x20); //function set

wait\_us(37);

write\_cmd(0x20); //function set

wait\_us(37);

write\_cmd(0x0C); //display ON/OFF

wait\_us(37);

write\_cmd(0x01); //display clear

wait\_us(1520);

write\_cmd(0x06); //entry-mode set

wait\_us(37);

write\_cmd(0x28); //function set

wait\_us(37);

}

void write\_4bit(int data, int mode){ //mode is RS line, cmd=0, data=1

int hi\_n;

int lo\_n;

hi\_n = (data & 0xF0); //form the two 4-bit nibbles that will be sent

lo\_n = ((data << 4) &0xF0);

//send each word twice, strobing the Enable line

shift\_out(hi\_n | ENABLE | mode);

wait\_us(1);

shift\_out(hi\_n & ~ENABLE);

shift\_out(lo\_n | ENABLE | mode);

wait\_us(1);

shift\_out(lo\_n & ~ENABLE);

}

void shift\_out(int data){ //Sends word to SPI port

CS = 0;

ser\_port.write(data);

CS = 1;

}

void write\_cmd(int cmd){ //Configures LCD command word

write\_4bit(cmd, COMMAND\_MODE);

}

void write\_data(char c){ //Configures LCD data word

write\_4bit(c, DATA\_MODE); //1 for data mode

}

void clr\_lcd(void){ //Clears display and waits required time

write\_cmd(0x01); //display clear

wait\_us(1520);

}

void print\_lcd(const char \*string){ //Sends character string to LCD

while(\*string) {

write\_data(\*string++);

wait\_us(40);

}

}

*Program Example 1: A two-task Program*

Enter the program into Mbed Studio or the on-line compiler, compile and download to your Nucleo board. You should see a count appearing on the LCD, while the on-board LED flashes. You will notice that the LED flashes only momentarily, rather than flashing evenly on and off, even though the program seems to require that. This is because we have inadvertently introduced a very simple resource clash, in this case the LED line. This was mentioned in an earlier Lab, when it was pointed out that the SPI port we’re using shares the line to the on-board LED. We will tolerate this for now.

## Introducing the Temperature Thread, and the Mutex

We now want to introduce the most important task, i.e. that of reading and displaying the temperature. (Here this is structured as a single task, although there may be an argument for splitting it into two.) This throws us a new challenge - we will have two tasks needing to access the LCD. That access we know is time-consuming, and likely to last well over a single RTOS time-slice. Hence we need a mutex, which will allow each task to use it exclusively until completed. As we would expect, this is available within the Mbed API.

Either extend your current program, or start a new one, and carefully add in the code fragments given in Program Example 2. Notice how the Mutex API is invoked, with **lcd\_mutex** as name of our choice. Apply also the mutex at start and end of the **count\_thread()** thread, following the pattern seen in **temp\_thread()**.

...

Mutex lcd\_mutex;

...

...

Thread thread3; //Read temp and display on LCD

void temp\_thread(void const \*args){

while(1) {

lcd\_mutex.lock();

//Write 0xAA to sensor to read the last converted temp

temp\_sensor.write(temp\_addr, &commands[1], 1, false);

//Read temp into the read\_temp array

temp\_sensor.read(temp\_addr, read\_temp, 2);

//Convert temp to Celsius. Shift MS byte right by 8 bits, OR in ls byte,

//then divide all by 256, with float result.

float temp = (float((read\_temp[0]<<8)|read\_temp[1])/256);

//convert float to character string

sprintf(LCD\_result, "%d.%d", int(temp), int((temp-int(temp))\*10));

write\_cmd(0x80); //set cursor to start of first line

wait\_us(40);

for (char i=0; i<4; i++) { //write 4 characters from string

write\_data(LCD\_result[i]);

} //end of for()

print\_lcd(" degrees C");

//Write 0x51 to the DS1631 (address 0x90) to start next temp conversion

temp\_sensor.write(temp\_addr, &commands[0], 1, false);

lcd\_mutex.unlock();

thread\_sleep\_for(1000);

}

...

...

thread3.start(callback(temp\_thread,&temp\_sensor));

...

*Program Example 2: The Temperature Sense Thread*

With your program complete, compiled and downloaded to your hardware, you should have a fully functioning temperature sensing unit, albeit without the final task we’re going to add.

## Adding the LED Brightness Thread

Now add a final thread, controlling the LED brightness from the potentiometer. You will need to define an analog input for the potentiometer, and a PWM output for the LED, for example using these lines:

...

PwmOut LED\_var(D3);

AnalogIn pot1(A0);

...

Then write a simple thread, which effectively transfers the value read on the analog input to the PWM output. As with others, make it an endless loop. Don’t forget to launch it from the **main()** function. Will you need to use a Mutex?

## Troubleshooting

Remember, it’s the experience of most embedded developers at some time to find yourself sitting and staring at a non-functioning new build, trying to work out if it’s the hardware or software or both that’s at fault! Now you’re engaging in more complex prototypes, you may find yourself in this situation. The simple troubleshooting tips given in Lab 0 are repeated here – check the points below with care:

* Are all circuit connections correctly made? Check that power and ground are correctly connected, and that LEDs are the right way round.
* Is the board and circuit powered correctly? If you have either, use a digital voltmeter or oscilloscope to check power distribution.
* Has the program compiled and downloaded properly? In the Nucleo, check for that little flickering bi-colour LED at the moment of download.
* Is your program logical, and does it really do what you’re hoping? Remember that a successful compile only tells you that the program is grammatically correct, not that it will lead to correct functioning.

If you have checked the above without success, return to a “known good” hardware and/or software version, and make incremental changes from there. “Known good” is likely to be the most recent working circuit that you have developed.

# Conclusion

If you have completed all the work of this Lab, then you’ve made a huge step forward in your program development skills. You have implemented use of an RTOS for the first time – congratulations! Of course you’re still new to the RTOS; there remains much to learn, and many more RTOS features to explore. Look out for the next Lab, to take you further in this exciting journey.

# References

*The five web pages below are significant landmarks of the on-line Mbed manual*

1. Introduction to ARM Mbed OS6

[Introduction - Introduction to Mbed OS 6 | Mbed OS 6 Documentation](https://os.mbed.com/docs/mbed-os/v6.6/introduction/index.html)

1. Full Mbed API listing

[Full API list - API references and tutorials | Mbed OS 6 Documentation](https://os.mbed.com/docs/mbed-os/v6.6/apis/index.html)

1. Mbed Tutorials and Examples

[Tutorials and official examples - Tutorials and examples | Mbed OS 6 Documentation](https://os.mbed.com/docs/mbed-os/v6.6/tutorials/index.html)

1. Mbed Components

[Components | Mbed](https://os.mbed.com/components/)

1. Mbed Forums

[Arm Mbed OS support forum - Get support for Arm Mbed OS from our community and support team](https://forums.mbed.com/)

*These books are excellent reference points while programming in C and/or C++*

1. Peter Prinz and Ulla Kirch-Prinz. (2002). *C Pocket Reference*. O’Reilly. ISBN 0-596-00436-2.
2. Kyle Loudon. (2003). *C++ Pocket Reference*. O’Reilly. ISBN 978-0-596-00496-5.