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

**LAB 2**

**Further Serial Communication**

1. Contents

[1 Introduction 1](#_Toc61879834)

[2 Resources 1](#_Toc61879835)

[3 Configuring the Circuit 2](#_Toc61879836)

[3.1 Reviewing I2C 2](#_Toc61879837)

[3.2 Introducing the DS1631 Temperature Sensor 2](#_Toc61879838)

[3.3 Connecting the Circuit 4](#_Toc61879839)

[4 Programming the Temperature Sensing Unit 6](#_Toc61879840)

[4.1 An Example Program 6](#_Toc61879841)

[4.2 Adding the LCD 8](#_Toc61879842)

[4.3 Taking Things Further 9](#_Toc61879846)

[4.3.1. Explore the accuracy of the measurement 9](#_Toc61879847)

[4.3.2. Explore the use of the DS1631 thermostat function 9](#_Toc61879848)

[4.3.3. Add another sensor to the I2C bus 9](#_Toc61879849)

[5 Conclusion 9](#_Toc61879850)

[6 References 9](#_Toc61879851)

# Introduction

This lab aims to produce a fully functioning temperature measurement system, with temperature displayed on LCD and computer screens. It applies the Inter-Integrated Circuit (I2C) serial protocol, as well as continuing to use the SPI and UART protocols introduced in Lab 1. The hardware builds directly on the circuit build of Lab 1, in other words, that circuit is our starting point. The particular objectives are:

* to introduce use of the I2C serial protocol;
* to introduce use of the DS1631 temperature sensor;
* to build and program a fully functioning temperature measurement system;
* to explore possible developments of this system.

# Resources

In this lab you can use either the Mbed Studio, or the on-line compiler (or both). Both were introduced in Lab 0 and used in Lab 1. When programming, remember the on-line and print references which were introduced in Lab 0; they are repeated here as References 2-8.

The hardware elements needed are listed in Table 1. Items which are in grey were used in Labs 0 or 1, and continue to be used. Those in black are new for this lab.

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

\*2 or more, if it is desired to develop a sensor network

*Table 1: List of Required Parts*

## 

# Configuring the Circuit

## Reviewing I2C

You will recall Figures 1 and 2 from the on-line material of this module. Figure 1 serves as a reminder that there are *only* two interconnecting wires on the I2C bus (apart from the ever-present ground and power interconnections). These are labelled **SDA** for data, and **SCL** for clock. Each wire must have a single pull-up resistor.

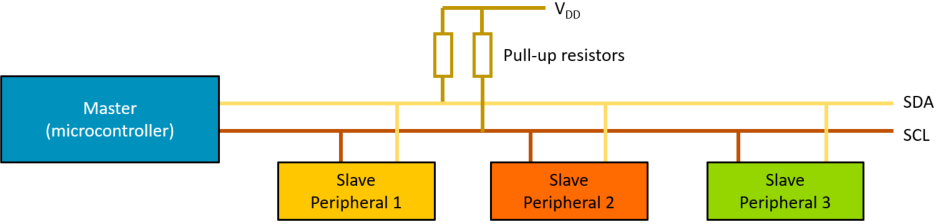


Figure 1: An I2C Bus Connection

The I2C data format, for a single data byte, is shown in Figure 2. Data transmission, initiated by the Master, starts with a Start condition. The first byte contains 7 bits of address, and a single R/|W bit, followed by an acknowledge bit from the receiver. The second byte is the data byte. More data bytes can be sent. The message is terminated by a Stop condition.

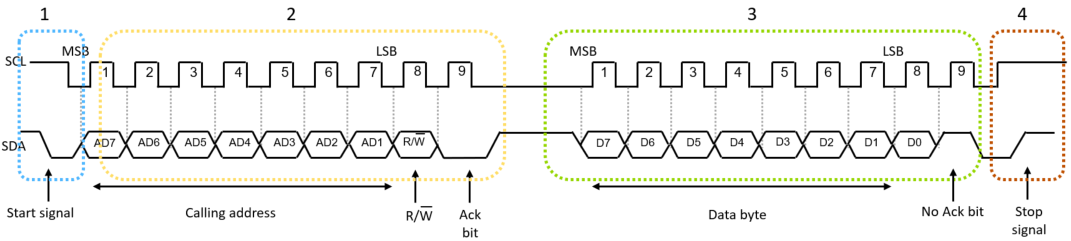


Figure 2: The I2C data format for a single byte of transmission.

## 3.2 Introducing the DS1631 Temperature Sensor

The package of the DS1631 temperature sensor is shown in Figure 3. Its full data sheet is found in Reference 1, and is well worth looking at. The sensor comes as an 8-pin integrated circuit, with the ability to act as an I2C slave. The device is made up of a semiconductor temperature sensor, an analog to digital converter, and a serial interface. It also contains a thermostat circuit, so the device can easily be used in simple temperature control applications.

|  |  |  |
| --- | --- | --- |
| Pin | Symbol | Description |
| 1 | SDA | Data Input/Output Pin for 2-Wire Serial Communication Port. Open Drain |
| 2 | SCL | Clock Input Pin for 2 Wire Serial Communication Port |
| 3 | TOUT | Thermostat Output Pin, Push and Pull |
| 4 | GND | Ground Pin |
| 5 | A1 | Address Input Pin |
| 6 | A2 | Address Input Pin |
| 7 | A3 | Address Input Pin |
| 8 | VDD | Supply Voltage Pin. (+2.7V to 5.5V) |

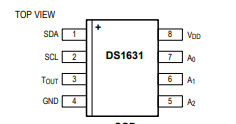


Figure 3: The DS1631 temperature sensor, Pin Connections and Pin Descriptions

The 7-bit address of the DS1631 will be inserted into the first byte of an I2C communication, i.e. byte 1 of Figure 2, repeated as Table 2a). It is formed as shown in Table 2a), i.e.by the 7-bit binary number 1 0 0 1 A2 A1 A0 . Thus by wiring the address pins (pins 5-7) of the DS1631 to Ground or VDD, the user can select any of 8 possible addresses. With the least significant byte (lsb) of the first message byte being R/|W, the slave address is often quoted with this lsb being set to 0. Hence the DS1631 address can be set to be any even number from 10010000 (0x90) to 10011110 (0x9e).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 1 | 0 | 0 | 1 | A2 | A1 | A0 | R/|W |

1. *The Address Byte of an I2C communication with the DS1631.*

|  |  |  |
| --- | --- | --- |
| Command | Command in Hex | Description |
| Start Convert T | 0x51 | Initiates temperature conversions |
| Stop Convert T | 0x22 | Stops temperature conversions when the device is in continuous conversion mode |
| Read Temperature | 0xAA | Reads the last converted temperature value from the 2-byte temperature register |
| Access TH | 0xA1 | Reads or writes the 2-byte TH register (used in Thermostat mode) |
| Access TL | 0xA2 | Reads or writes the 2-byte TL register (used in Thermostat mode) |
| Access Config | 0xAC | Reads or writes the 1-byte configuration register |
| Software POR | 0x54 | Initiates a software power-on-reset (POR), which stops temperature conversions and resets all registers and logic to their power-up states. |

b): The DS1631 Command Set

*Table 2*

Table 2b) shows the commands which are used to control the DS1631, and is reasonably self-explanatory. The result of each temperature measurement is held in the 16-bit temperature register. The data is “left aligned” i.e. it occupies the full 8 bits of the more significant byte, and the three most significant bits of the less significant byte. The data format is “two’s complement”, which allows easy representation of a temperature below 0o C. However we are likely only to be dealing with positive temperatures in this application.

In this introductory experiment we use the DS1631 sensor in a simple way. There is a configuration register, but we make no adjustments to its default settings. Nor do we make use of the facility to check if a conversion is complete, simply waiting for a fixed and possibly unnecessarily long period to allow completion.

## 3.3 Connecting the Circuit

The circuit for the Temperature Sensing Unit is shown in Figure 4. It’s easy to see that this builds directly on the final circuit of Lab 1, with the fairly straightforward DS1631 temperature sensor sub-circuit being added. Address pins 5, 6 and 7 of the DS1631 are connected to ground, which means that the temperature sensor address will be 1001000, or 0x90. According to I2C requirements, two pull-up resistors are connected on the SDA and SCL. Values of 1kΩ are used for this prototype; for a final design this value could be optimised.

Connect up the circuit of Figure 4. If by chance you’re building from scratch, you can omit the LCD at this stage, if you wish. The connection details of the Nucleo board are included in Figure 5 for convenience. A completed build is shown in Figure 6.

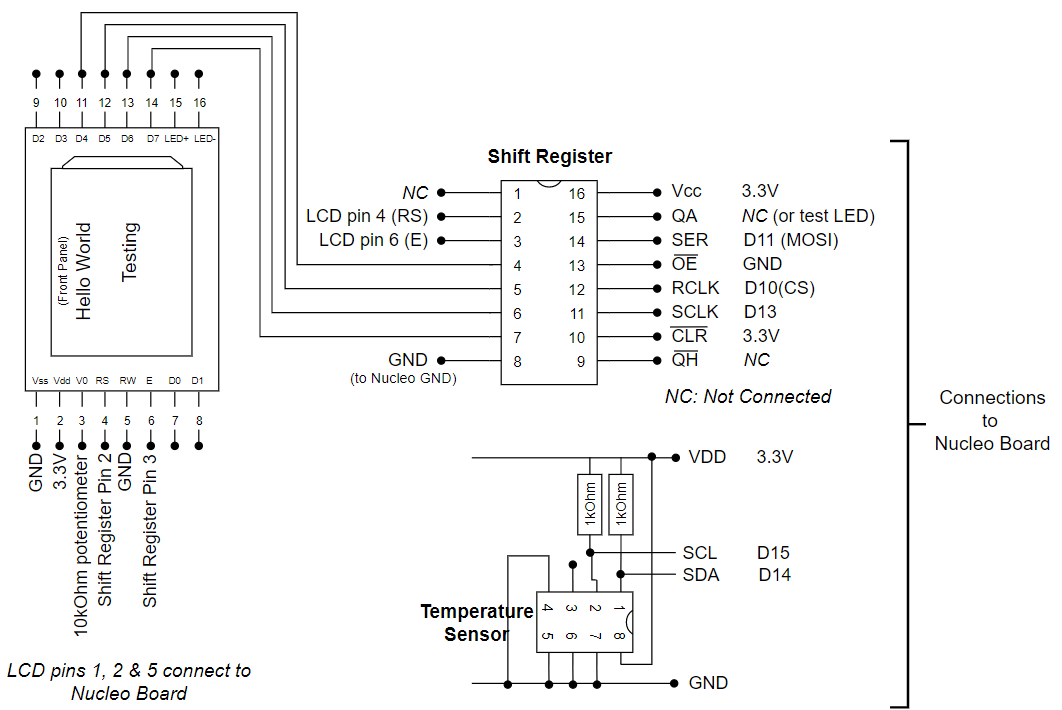


Figure 4: Circuit Layout

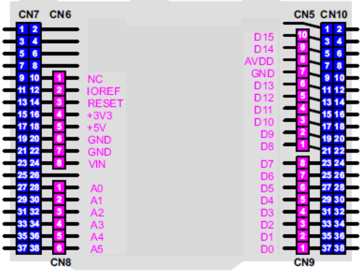


Figure 5: Connections on Nucleo Board, Arduino Connectors Only

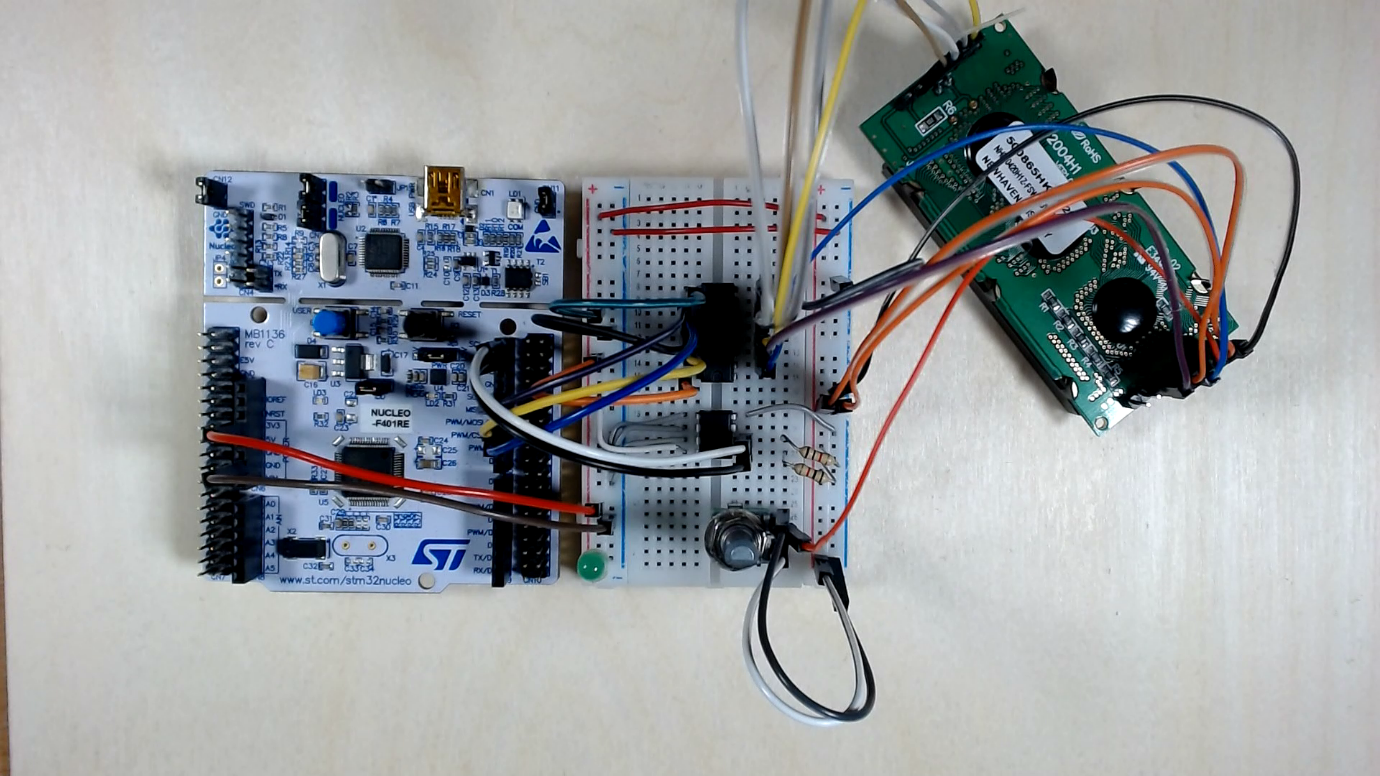


Figure 6: Completed Build of the Temperature Sensor Circuit

# Programming the Temperature Sensing Unit

## An Example Program

The program for the circuit needs to:

* Write the Start Convert command to the sensor;
* Wait for the conversion to complete, then write the Read Temperature command to the sensor;
* Read the 16-bit temperature data;
* Convert the binary temperature data into decimal;
* Print the temperature to the PC via the UART (note that the LCD will be implemented later).

The member functions of the Mbed I2C API, as shown in Table 3, will be applied to make the data transfers, and the DS1631 commands of Table 2 to control the temperature sensor.

|  |  |
| --- | --- |
| Function name | Description |
| I2C (PinName sda, PinName scl) | Create an I2C Master interface, connected to the specified pins |
| void frequency (int hz) | Set the frequency of the I2C interface |
| int read (int address, char \*data, int length, bool repeated=false) | Read from an I2C slave |
| int read (int ack) | Read a single byte from the I2C bus |
| int write (int address, const char \*data, int length, bool repeated=false) | Write to an I2C slave |
| int write (int data) | Write single byte out on the I2C bus |
| void start (void) | Creates a start condition on the I2C bus |
| void stop (void) | Creates a stop condition on the I2C bus |

*Table 3: Member Functions of the Mbed I2C API*

Enter the program into Mbed Studio or the on-line compiler, and compile and download it to your Nucleo board.

/\*Reads temp from DS1631 and displays on PC screen

\*/

#include "mbed.h"

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];

int main(){

while(1){

//Write 0x51 to 0x90 to start temperature conversion

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

thread\_sleep\_for (500); //wait for the conversion to complete

//Write 0xAA to 0x90 to read the last converted temperature

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

//Read the temperature into the read\_temp array

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

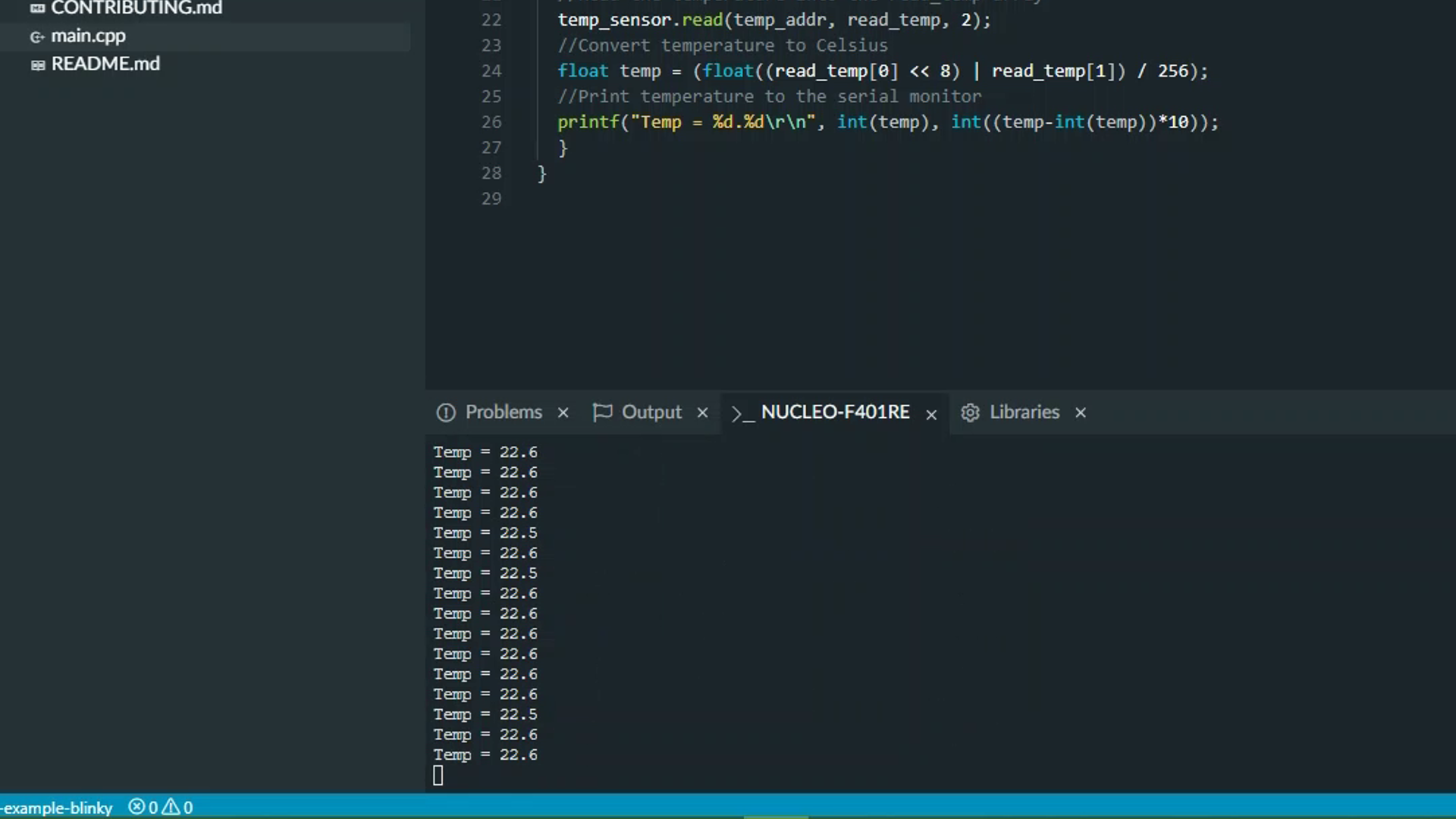
printf("Temp = %d.%d\r\n", int(temp), int((temp-int(temp))\*10));

}

}

*Program Example 1: Reading Temperature and Displaying on PC Screen*

With Program Example 1 running, and a serial terminal open on the PC screen, output such as that shown in Figure 7 should be displayed.



*Figure 7: Mbed Studio Terminal Output from the Temperature Sensor Unit*

## Adding the LCD

## We now want to develop Program Example 1 so that data is displayed on the LCD as well. Create a new project in Mbed Studio or the On-line Compiler, and copy into it Program Example 1 from this Lab. Then go back to Program Example 2 in Lab 1, and copy into your new program all the functions relating to the LCD, with function prototypes, and all other preliminaries needed for the LCD to function correctly (for example, check the #define statements). A close comparison of Program Example 2 in Lab 1 with Program Example 1 from this Lab will help you with this.

## Now we need to extend the main() function, to include writing to the LCD, both fixed text and the variable result. There are Mbed library functions which do this for us, but for now we’re aiming not to use them, in order to get some more programming practice. Our extra bit of code will need to format the data into decimal characters, and then get these characters sent to the LCD using the functions we already have. A useful standard C/C++ function is sprint() , which converts a float variable into a character string, with decimal point thrown in if needed.

## Either devise your own method, or take the lines in Program Example 2 and add them to your new program, in the places indicated.

//Insert this line with the other variable declarations

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

...

...

...

//Insert these lines at the very start of Main()

init\_lcd(); //initialise the LCD

clr\_lcd(); //Clear the LCD

print\_lcd("Temperature is");

...

...

...

//Insert these lines at the end of the While(1) loop in Main()

//convert float value in temp to character string in LCD\_result

sprintf(LCD\_result, "%d.%d", int(temp), int((temp-int(temp))\*10)); write\_cmd(0xc0); //set cursor to start of second line

wait\_us(40);

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

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

}

print\_lcd(" degrees C"); //add units

...

...

*Program Example 2: Code Snips to Add to Modify Existing Program or add to a New Program*

When you have put your new program together, compile it and download to your hardware. This should result in a very pleasing system, which measures temperature and displays both on the PC screen and the LCD.

## Taking Things Further

## 4.3.1. Explore the accuracy of the measurement. Study the DS1621 carefully and determine how the measurement could be adjusted (if at all) to be:

* Quicker, with the same accuracy?
* Quicker, with reduced accuracy?
* Slower, with greatest accuracy?

For each of these, how valid is the on-screen display, which implies accuracy to two decimal places?

## 4.3.2. Explore the use of the DS1631 thermostat function. Set thresholds close to your current room temperature, and monitor the TOUT output of the sensor. Can you connect an LED to this, and/or interface it with the Nucleo board?

## 4.3.3. (Requires a second temperature sensor). Add another sensor to your I2C bus, ensuring it has a different slave address from the first. Display both temperatures on the PC screen and/or LCD. With careful wiring, you could place the sensors at two different places in the room, and monitor the difference.

# Conclusion

You have experienced the development of a useful prototype electronic product, in the form of the temperature sensing unit. If you have attempted some of the challenges offered in Section 4.3, then you will have experienced some genuine independent development work. Keep working on your programming fluency, and circuit development skills.

You have been developing your programming skills using conventional sequential programming. In the next two labs we develop a new style of programming, explicitly applying the Real Time Operating System (RTOS).

# References

1. DS1631 temperature sensor datasheet

[http://datasheets.maximintegrated.com/en/ds/DS1631-DS1731.pdf](about:blank)

*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.