Sheffield Hallam University
Department of Engineering
BEng (Hons) Electrical and Electronic Engineering

Activity ID		Activity Title			Laboratory Room No.	Level
Lab 102		Using sensors with Arduino			4302	6
Semester	**Duration [hrs]**	**Group Size**	**Max Total Students**	**Date of approval**	**Lead Academic**	
1	2	1	25	09-22	Alex Shenfield	

Equipment (per student/group)

Number	Item
1	PC
1	Arduino kit

Learning Outcomes

	Learning Outcome
2	Demonstrate an understanding of the various tools, technologies and protocols used in the development of embedded systems with network functionality
3	Design, implement and test embedded networked devices

Using sensors with Arduino

Introduction

Computing is about more than the PC on your desktop! Embedded devices are everywhere – from wireless telecommunications infrastructure points to electronic point of sale terminals. One definition of an embedded system is:

> "**An embedded system is a computer system designed to perform one or a few dedicated functions** often with real-time computing constraints."
> (http://en.wikipedia.org/wiki/Embedded_system)

In this series of labs you are going to be introduced to the open source Arduino platform – a cheap, simple to program, well documented prototyping platform for designing electronic systems. Although the Arduino platform started out with the Arduino Uno (based on the commonly used ATMega328 chip), other, more capable, boards have since been released. In this series of labs you will be using the Arduino MKR WiFi 1010 board – a Cortex M0+ based Arduino board with built in WiFi module.

So far in the laboratory sessions we have provided a fairly gentle introduction to both the software and hardware aspects of the Arduino platform as well as some of the basic electronics you'll need to create functional embedded systems. In this laboratory session we are going to deal with an extremely important aspect of any kind of embedded system – **input to the system.**

Any kind of embedded microcontroller is only really useful if it has some ability to see the world – whether as simple as taking input from switches and potentiometers or as complicated as reading and reacting to accelerometer data. To do this we will need to integrate sensors into our system.

By the end of this set of laboratory exercises you will understand how to read digital, analog, and serial sensor data with the Arduino MKR WiFi 1010 and process that data to effect the world in some way.

Bibliography

There are no essential bibliographic resources for this laboratory session aside from this tutorial sheet. However the following websites and tutorials may be of help (especially if you haven't done much electronics previously or your digital logic and/or programming is a bit rusty):

- http://www.arduino.cc/
- http://www.ladyada.net/learn/arduino/index.html
- http://tronixstuff.wordpress.com/tutorials/

Methodology

Check that you have all the necessary equipment (see Figure 1)!

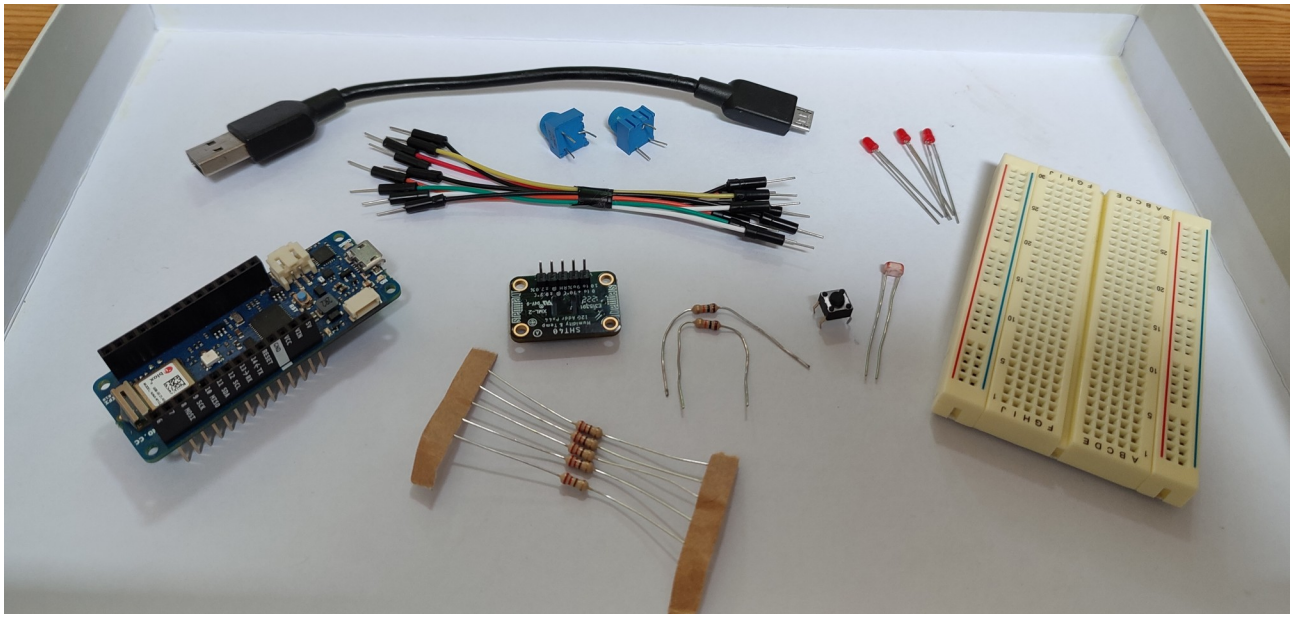

Figure 1 – The necessary equipment for this lab (don't worry if you have LEDs of a different colour or slightly different breadboards)

Task 1

In this task we are going to use a simple push-button switch to control an LED.

Firstly create the circuit shown in Figure 2, below. This wires up the two upper rails of the breadboard to the +5V and ground pins on the Arduino board (+5V to the lower rail and ground to the top rail) allowing us to easily access the +5V supply and ground. The push-button switch then has one leg connected to +5V and the other leg connected to ground via a pull-down resistor (in this case a 220 Ohm one). The other side of this leg is connected to digital pin 12 on our Arduino board. We then connect digital pin 13 to the LED.

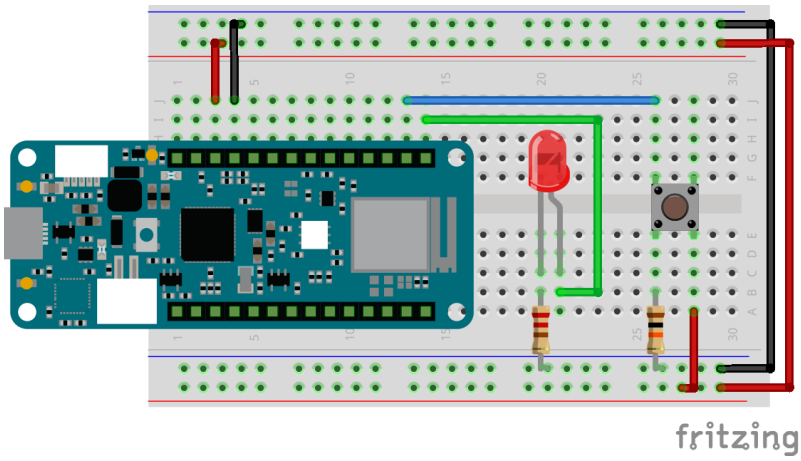

Figure 2 – The switch circuit

Q Why do we need a "pull-down" resistor?

Now we wish to write some code to control this circuit. When the push-button is held down we want the LED to turn on, and then when the push-button is released we want the LED to turn off again. See code listing 1 for a simple program to accomplish this.

Code listing 1: LED on/off code

```
/*
 * 1_arduino_switch.ino
 *
 * this is a simple Arduino program to use a push button (i.e. a momentary
 * switch) to turn on an led
 *
 * this program uses pin 7 for the push button and pin 6 for the led (this
 * is the built in led on the MKR series of boards).
 *
 * author: alex shenfield
 * date:   01-09-2022
 */

// the button is connected to pin 7 and the led is connected to pin 6
const int button_pin = 7;
const int led_pin = 6;

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // sets the pin directions
  pinMode(button_pin, INPUT);
  pinMode(led_pin, OUTPUT);
}

// this methods loops continuously
void loop()
{
  // if the button is pressed, turn the led on
  int button_state = digitalRead(button_pin);
  if(button_state == HIGH)
  {
    digitalWrite(led_pin, HIGH);
  }
  else
  {
    digitalWrite(led_pin, LOW);
  }
}
```

Enter this program and make sure everything works. If there are any problems contact a member of staff.

Exercises

Now you are going to make some modifications to the program to add functionality:

a. Alter the above program to "latch" the button state (i.e. pressing the button turns the LED on, pressing it again turns the LED off). To do this you will need to declare another variable that keeps track of the last state of the LED (i.e. whether it was last on or off).

b. You may have noticed that, in your "latched" button state program, the LED sometimes doesn't behave as expected – this is because the mechanical contacts on the switch may "bounce", triggering the latch in quick succession. Figure 3 shows an oscilloscope trace of a button being pressed.

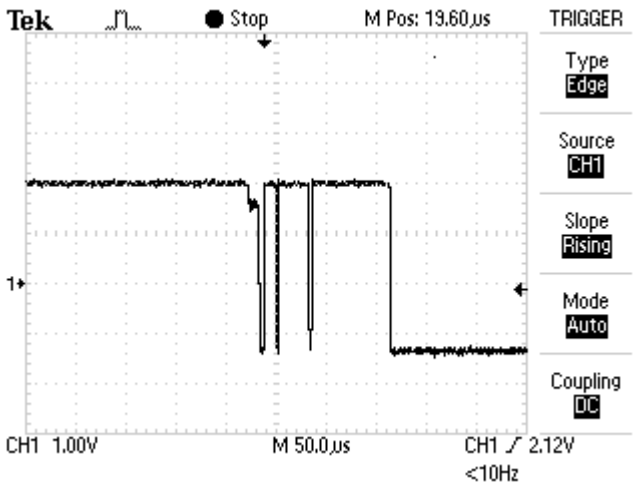

Figure 3 – Oscilloscope trace of a button press

As you know, switch debouncing can be done either in hardware (using a simple low pass filter) or software. Adding debouncing in hardware at design time is usually the best solution – however, sods law means that often these problems don't make themselves apparent until after production has started! Then, instead of the costly process of remaking all the hardware, we can fix these problems in software instead!

You should add "debouncing" to your program (which means checking the button state twice – or more – in a short period of time to ensure that the button was actually pressed and it wasn't just the contacts "bouncing"). Some more information can be found at:

http://www.labbookpages.co.uk/electronics/debounce.html
https://www.arduino.cc/en/Tutorial/Debounce

Ask a member of staff is this is still unclear.

c. Wire up two other LEDs of different colours. Adapt your program to cycle through the LEDs when the button is pressed.

Task 2

In this task we are going to read the ambient light value using a Light Dependent Resistor (LDR) and display it in the Serial Monitor. An LDR is a passive component where the resistance varies according to the light hitting its active area (the brown material – usually cadmium sulphide – on the top of the component). Passive components do not amplify or add energy to the circuit.

In this task we will create a resistor-divider circuit using the LDR and a 10K ohm resistor (so that the change in resistance of the LDR results in a change of voltage at the analog input pin of the Arduino). Here the LDR has one leg connected to VCC (on the Arduino MKR WiFi 1010 this is 3.3V) and the other to A0. The 10K ohm resistor is then connected between this leg and ground. This is shown clearly in Figure 4.

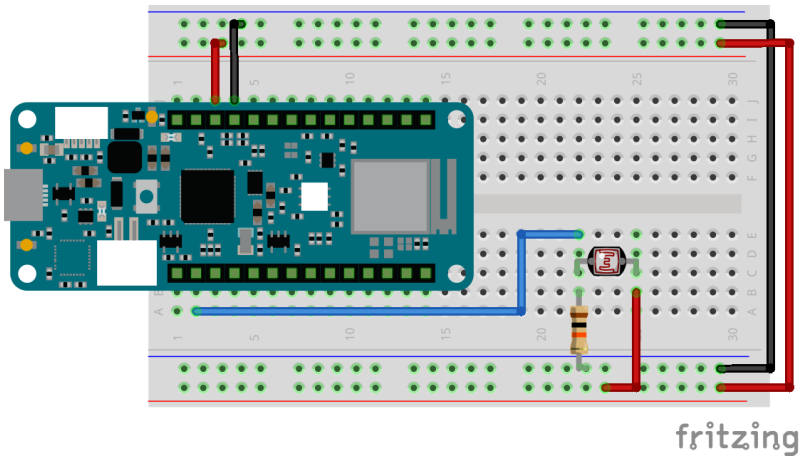

Figure 4 – The LDR circuit

Code listing 2 then shows a simple program to read this voltage value and display it on the Arduino IDE Serial Monitor.

Code listing 2: Code to read the LDR value and display it on the Serial Monitor

```
/*
 * 2_analog_sensor.ino
 *
 * this is a simple Arduino program to read from an analog sensor (in this
 * case an ldr) and write the sensor reading to the serial monitor
 *
 * this program uses pin A0 for the analog sensor
 *
 * author: alex shenfield
 * date:   01-09-2022
 */

// the ldr is connected to A0
const int ldr_pin = A0;

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // initialise serial communication at 115200 bits per second, wait for
  // the serial port to become available, and print a start up message
  Serial.begin(115200);
  while (!Serial)
  {
    delay(1);
  }
  Serial.println("analog sensor example");
}

// this methods loops continuously
void loop()
{
  // read the analog sensor and print to the serial monitor
  int ldr_value = analogRead(ldr_pin);
  Serial.println(ldr_value);

  // delay for 100 ms between readings
  delay(100);
}
```

Enter this program, upload it to the board, and make sure everything works. When you access the Serial Monitor (the magnifying glass in the top right corner of the IDE), you should see data displayed similar to that shown in Figure 5.

Figure 5 – Light data displayed on the Arduino Serial Monitor

Note the use of Serial.println() for keeping track of the value of our LDR in this sketch by logging the analog value read from the LDR pin (pin A0) to the serial monitor. You will find that this functionality is extremely useful for debugging your code when writing programs of any complexity!

A neat trick here is that we can use the "Serial Plotter" built in to the Arduino IDE[1] to graph these light values (see Figure 6)!

Figure 6 – Plotting light data via the Arduino Serial Plotter

1 This can be found under "Tools > Serial Plotter", or accessed via "Ctrl + Shift + L".

Exercises

Now you are going to make some modifications to the system to add functionality:

a) Add an LED into your system and create a new program that makes the LED light up when the light value falls below a certain threshold. You have now built a smart light switch!

b) Add more LEDs to your system and create a light sensitive LED bar graph (see Figure 7 for an example).

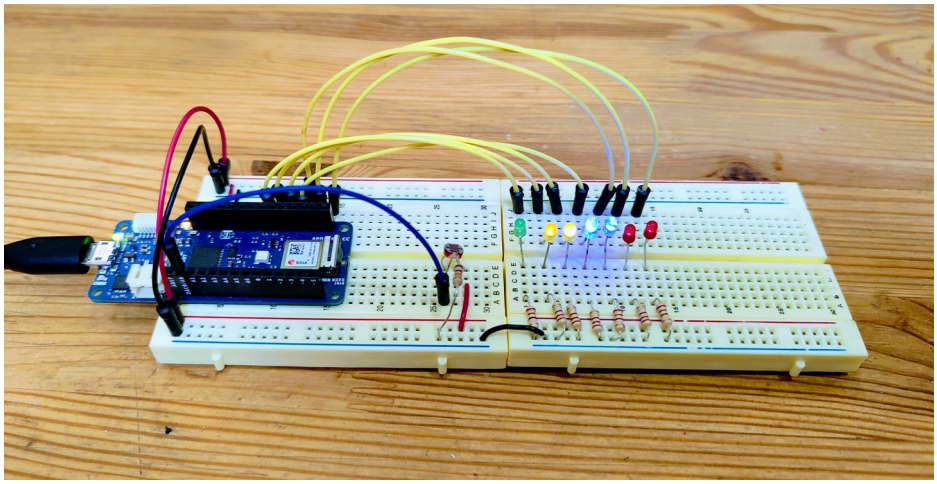

Figure 7 – An example light sensitive LED bar graph

Task 3

In this task we are going to read an I2C temperature and humidity sensor (the Adafruit Sensirion SHT40 Temperature & Humidity Sensor[2] - see Figure 8) and display the data in the Serial Monitor. The SHT40 has ±1.8% typical relative humidity accuracy from 25 to 75% and ±0.2 °C typical temperature accuracy from 0 to 75 °C. It is important to note that the I2C address for this sensor is 0x44 and cannot be changed[3].

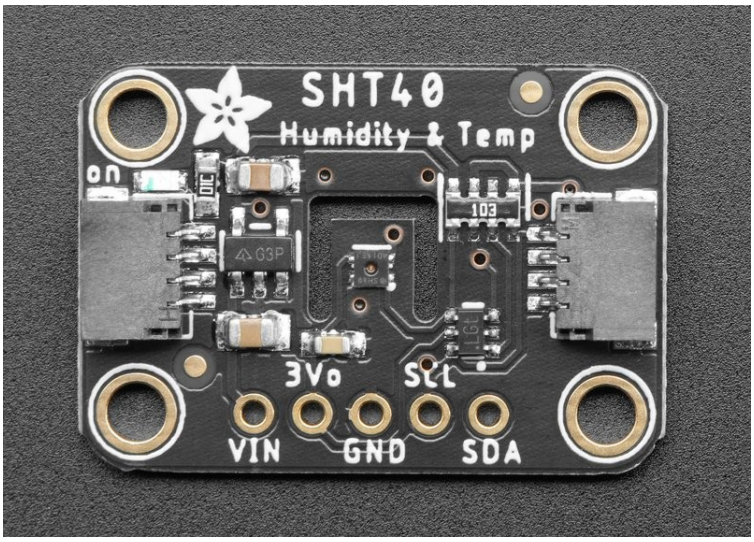

Figure 8 – The Adafruit Sensirion SHT40 Temperature & Humidity Sensor

The pin outs of this board are:

Pin	Description
VIN	This is the power pin – supply the same voltage as the logic level of your microcontrolle (i.e. 3.3V for the MKR WiFi 1010.
3Vo	This is the 3.3V output from the voltage regulator.
GND	Ground.
SCL	I2C clock line (note: there is an inbuilt 10K pull-up resistor on this line).
SDA	I2C data line (note: there is an inbuilt 10K pull-up resistor on this line too).

2 https://learn.adafruit.com/adafruit-sht40-temperature-humidity-sensor
3 So, as mentioned in the I2C lecture, if you have other I2C devices that use the same address you will probably run into issues!

Wire the sensor up to the Arduino MKR WiFi 1010 as shown in Figure 9. You can see on the side of the Arduino board there are dedicated pins for I2C communication (pins 11 and 12). Remember, I2C is a two wire protocol so all I2C devices are attached to common SCL and SDA connections.

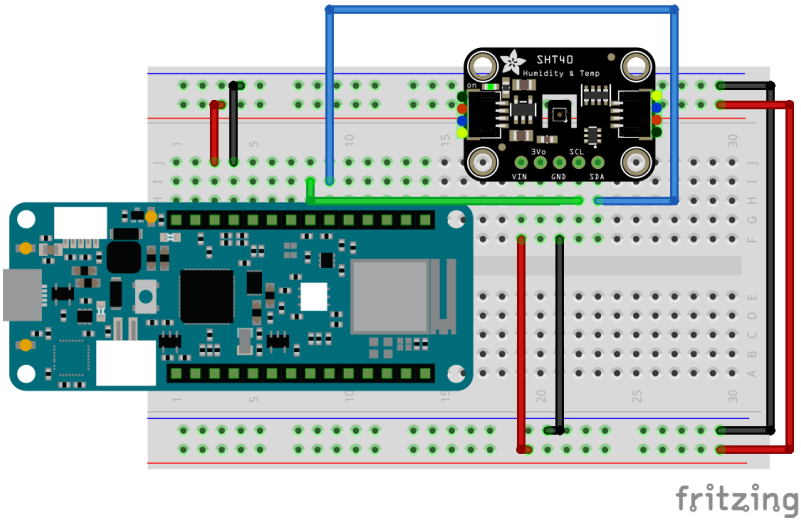

Figure 9 – SHT40 and Arduino MKR WiFi 1010 circuit

Before programming the board, we will need to install the Adafruit SHT4X Library using the Arduino IDE Library Manager (see Figure 10).

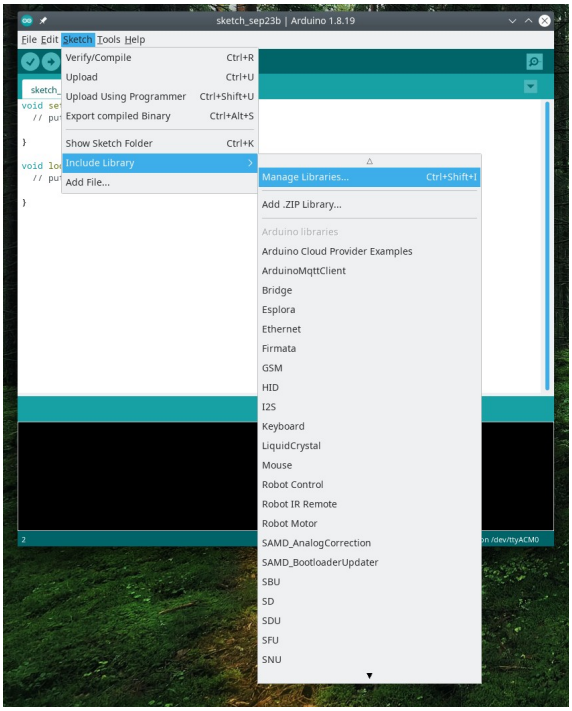

Figure 10 – Accessing the Arduino IDE Library Manager

Once you have clicked the “Manage Libraries ...” menu item, search for “SHT4X”, and click the “Install” button for the Adafruit SHT4X Library (see Figure 11).

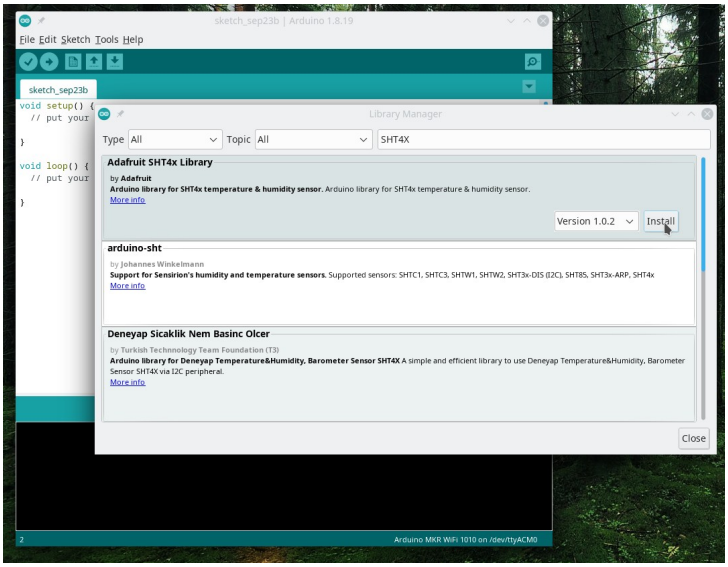

Figure 11 – Installing the Adafruit SHT4X Library

You will probably then see a message about dependencies for the Adafruit SHT4X Library (see Figure 12). Just click “Install all” to install these alongside the Adafruit SHT4X Library.

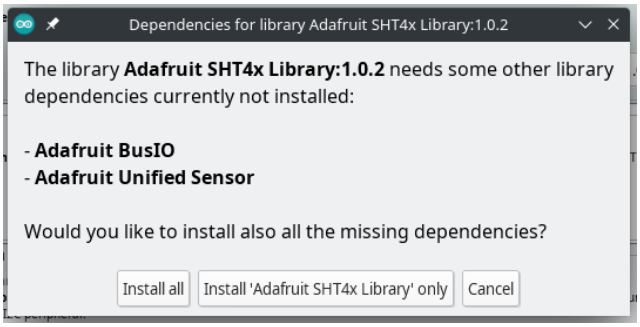

Figure 12 – Installing library dependencies

Now we need to write some code to configure and read the sensor. The SHT40 has several levels of precision / repeatability we can configure when reading the sensor, and a variable power heating element (useful for removing condensed water from the surface of the sensor to ensure more accurate readings in high humidity environments). See code listing 3 for a simple program to configure and read the sensor for normal ambient conditions.

Code listing 3: Code to configure and read the SHT40 sensor and display the values on the Serial Monitor

```
/*
 * 3_reading_an_i2c_temperature_and_humidity_sensor.ino
 *
 * this is a simple Arduino program to read from an i2c sensor (in this
 * case an SHT40 temperature and humidity sensor) and write the sensor
 * reading to the serial monitor
 *
 * this program uses pins 11 and 12 for SDA and SCL respectively (these
 * are the default i2c pins on the Arduino MKR WiFi 1010)
 *
 * author: alex shenfield
 * date:   01-09-2022
 */

// we will use the adafruit SHT4X library to simplify use of this
// sensor
#include "Adafruit_SHT4x.h"

// create the sensor object
Adafruit_SHT4x sht40 = Adafruit_SHT4x();

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // initialise serial communication at 115200 bits per second, wait for
  // the serial port to become available, and print a start up message
  Serial.begin(115200);
  while (!Serial)
  {
    delay(1);
  }
  Serial.println("i2c sensor example");

  // start the SHT40 sensor
  if (!sht40.begin())
  {
    // if it doesn't connect, print an error message and wait forever
    Serial.println("couldn't find SHT40 sensor - check connections!");
    while (true)
    {
      delay(1);
    }
  }

  // print the serial number of the sensor (for information)
  Serial.println("found SHT40 sensor");
  Serial.print("serial number 0x");
  Serial.println(sht40.readSerial(), HEX);

  // configure the sensor to have high precision (i.e. high repeatability
  // and heater off - see lab sheet for description of when using the
  // heater is useful
  sht40.setPrecision(SHT4X_HIGH_PRECISION);
  sht40.setHeater(SHT4X_NO_HEATER);

}
```

```
// this methods loops continuously
void loop()
{
  // initialise structures to contain the sensor readings
  sensors_event_t humidity;
  sensors_event_t temp;

  // read the sensor (and keep track of how long it takes)
  long timestamp = millis();
  sht40.getEvent(&humidity, &temp);
  timestamp = millis() - timestamp;

  // print temperature
  Serial.print("temperature: ");
  Serial.print(temp.temperature);
  Serial.println(" degrees C");

  // print humidity
  Serial.print("humidity: ");
  Serial.print(humidity.relative_humidity);
  Serial.println("%");

  // print the sensor read time
  Serial.print("reading took ");
  Serial.print(timestamp);
  Serial.println(" ms");

  // we'll use a simple 1s delay here because our loop is simple - if
  // we wanted to do anything more complicated in our loop, we'd be
  // better off not using delays (because they mean our program can't
  // do anything else)
  delay(1000);
}
```

Enter this program, upload it to the board, and make sure everything works. When you access the Serial Monitor, you should see data displayed similar to that shown in Figure 13.

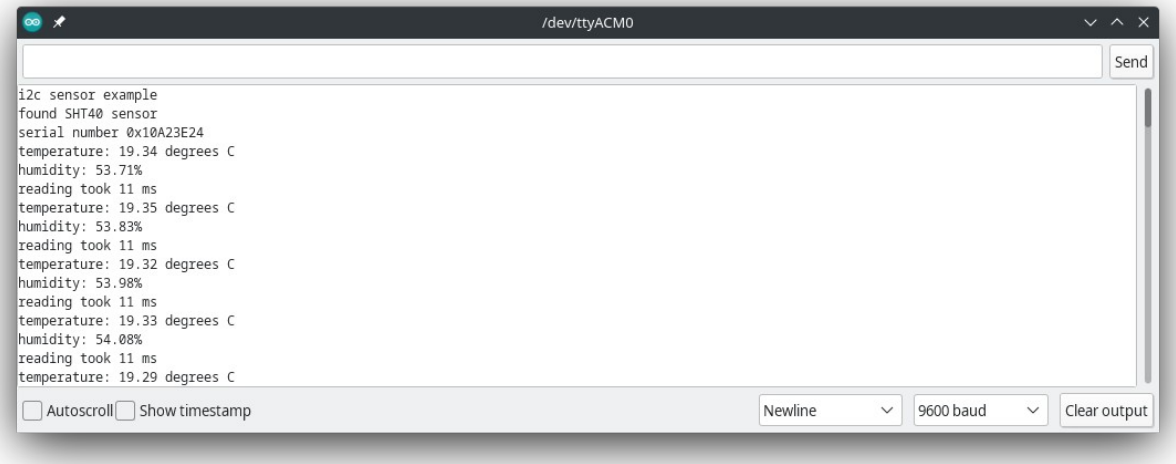

Figure 13 – Display of temperature and humidity data

Exercises

Now you are going to make some modifications to the system to add functionality:

a) Add some indicator LEDs to light up when the temperature and humidity values go above some preset threshold values.

b) Add two potentiometers to set these threshold values. The Ardunio MKR WiFi 1010 analog to digital convertors read from 0 – 1023, so you will have to remap these values to a sensible range (e.g. 5 – 30 for temperature and 0 – 100 for relative humidity). You can do this by using the "map" command in the Arduino language:

https://www.arduino.cc/reference/en/language/functions/math/map/

c) Now add the "smart" light sensor functionality you developed in task 2a) to your system. You now have the basis of a building environment monitoring system.

Check list

Task	Completed
Task 1 – Program	
Task 1 – Question	
Task 1 – Exercise a)	
Task 1 – Exercise b)	
Task 1 – Exercise b)	
Task 2 – Program	
Task 2 – Exercise a)	
Task 2 – Exercise b)	
Task 3 – Program	
Task 3 – Exercise a)	
Task 3 – Exercise b)	
Task 3 – Exercise c)	

Feedback