Sheffield Hallam University

Department of Engineering

BEng (Hons) Computer Systems Engineering

Activity ID		Activity Title			Laboratory Room No.	Level
Lab 101		An introduction to Arduino			4302	6
Semester	**Duration [hrs]**	**Group Size**	**Max Total Students**	**Date of approval/review**	**Lead Academic**	
1	4	1	25	09-19	Alex Shenfield	

Equipment (per student/group)

Number	Item
1	PC
1	Arduino kit

Learning Outcomes

	Learning Outcome
1	Understand the benefits, challenges and pitfalls of developing internet enabled embedded systems.
3	Implement multi-component distributed embedded systems using a flexible network architecture.

An introduction to 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)

The purpose of this introductory lab session is to introduce the Arduino Uno board and show you how to set up the development environment. You will then develop a simple Arduino program to blink a single LED (this is the embedded electronics version of Hello World!) and learn a bit about the programming language the Arduino platform uses. After that we will take a quick tour through the core input and output functionality that the Arduino platform offers.

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/

Task 1

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

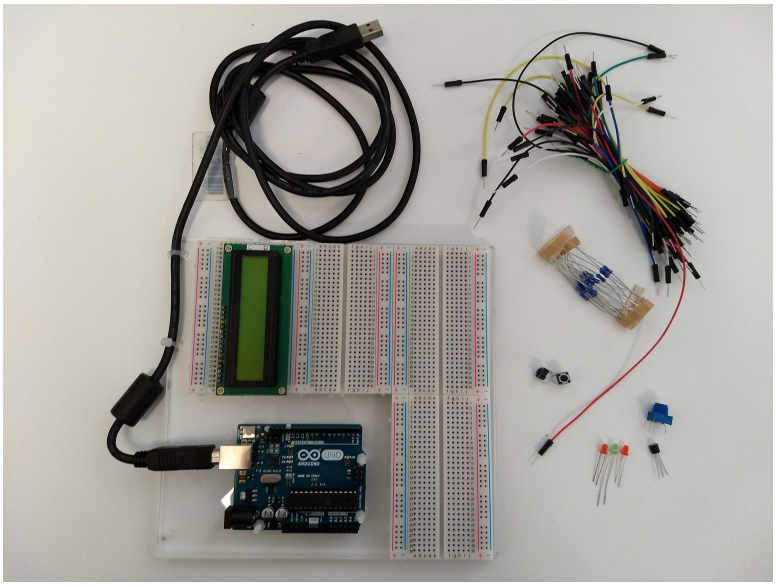

Figure 1 – The necessary equipment for this lab (don't worry if you have LEDs of a different colour)

2. Open the Arduino IDE by double clicking the Arduino application in the Arduino folder or searching on the Start Menu.

3. Select the correct board (Arduino Uno) and serial port (see Figures 2 and 3).

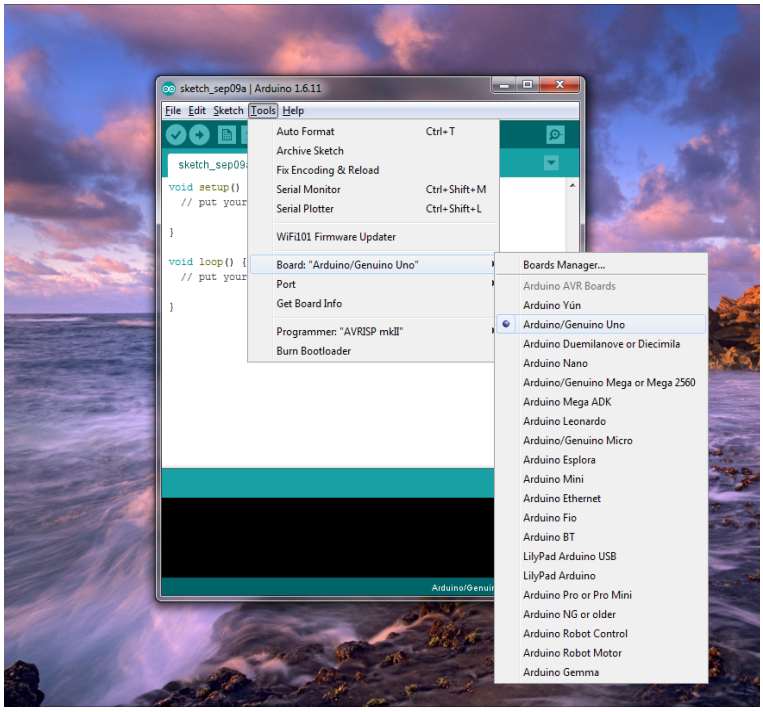

Figure 2 – Selecting the correct Arduino board

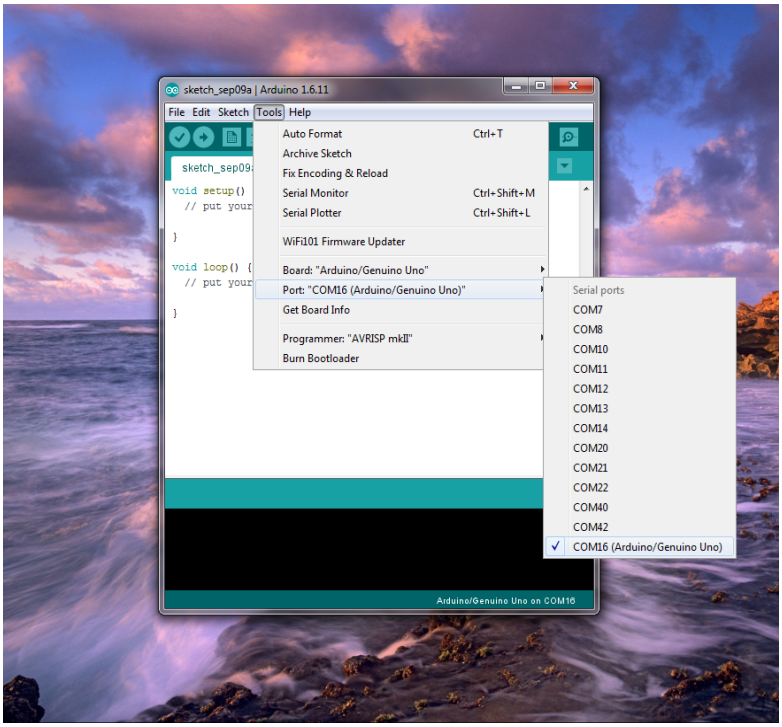

Figure 3 – Selecting the serial port

4. Enter the program shown in code listing 1 below (in Arduino parlance this is called a “sketch”). Appendix A has a full walk through of this code – so if there is anything that is unclear, refer to Appendix A (or ask!).

Code Listing 1: The Blink sketch

```
/*
 * blink.ino
 *
 * this is a simple Arduino program to blink an led (the electronics
 * equivalent of hello world). this program turns an led on for 1
 * second and then off for 1 second, and so on ...
 *
 * this program uses pin 13 for the led
 */

// the led is connected to pin 13
const int led_pin = 13;

// this method runs once (when the sketch starts)
void setup()
{
  // sets the digital pin as output
  pinMode(led_pin, OUTPUT);
}

// this methods loops continuously
void loop()
{
  // turn led on, wait for 1 second, turn led off, wait for 1 second,
  // repeat ...
  digitalWrite(led_pin, HIGH);
  delay(1000);
  digitalWrite(led_pin, LOW);
  delay(1000);
}
```

5. Verify / compile the sketch (by pressing the “tick” icon in the toolbar of the Arduino IDE) to make sure that it is correct.

6. Upload the sketch to the Arduino board (by pressing the right facing arrow icon in the toolbar of the Arduino IDE).

7. Insert the LED (as in Figure 4).

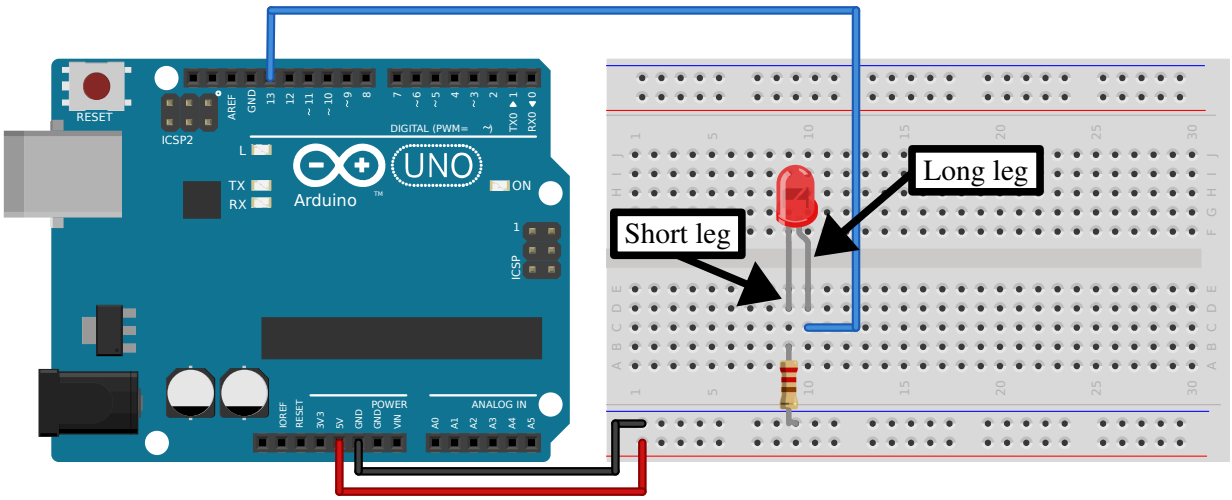

Figure 4 – Basic “blink” circuit

Task 2

Sometimes you need to do two things at once. For example you might want to blink an LED (or some other time-sensitive function) while reading a button press or other input. In this case, you can't use delay(), or you'd stop everything else the program while the LED blinked. The program might miss the button press if it happens during the delay(). This sketch demonstrates how to blink the LED without using delay(). It keeps track of the last time the Arduino turned the LED on or off. Then, each time through the loop() method, it checks if a long enough interval has passed. If it has, it toggles the LED on or off (depending on it's previous state).

Firstly we will set up the same LED circuit as previously (see Figure 4, previously).

Now we can use the program in code listing 2 to implement the blink sketch without a delay function.

Code Listing 2: Blink without a delay

```
/*
 * blink_without_delay.ino
 *
 * this is a simple Arduino program to blink an led (the electronics
 * equivalent of hello world) but without using the delay statement.
 * this program turns an LED on for 1 second and then off for 1
 * second, and so on ...
 *
 * this program uses pin 13 for the led
 */

// the led is connected to pin 13
const int led_pin = 13;

// state machine variables ...
long previous_time = 0;          // the last time the LED was updated
long on_interval = 1000;         // the on interval
long off_interval = 1000;        // the off interval
int  next_ledstate = 1;          // initial state of LED (on)

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  pinMode(led_pin, OUTPUT);      // sets the digital pin as output
}
```

```
// this methods loops continuously
void loop()
{
  // get the current time this time round the loop
  unsigned long current_time = millis();

  // state machine 1 - state (1), led on
  if((next_ledstate == 1) && (current_time - previous_time >= on_interval))
  {
    // next state ...
    next_ledstate = 2;

    // save the time
    previous_time = current_time;

    // turn led on
    digitalWrite(led_pin, HIGH);
  }

  // state machine 1 - state (2), led off
  else if((next_ledstate == 2) && (current_time - previous_time >= off_interval))
  {
    // next state ...
    next_ledstate = 1;

    // save the time
    previous_time = current_time;

    // turn led off
    digitalWrite(led_pin, LOW);
  }
}
```

This pattern is known as a **state-machine** (because we are storing some variables relating to the **state** of the led – i.e. whether the led is currently on or off and when it last changed – and some variables relating to the **transition** between states – i.e. the on interval and off interval).

Enter and test this code.

Exercises

- Try changing the on / off interval times.
- Try adding a new state machine (with different transition times) for an additional LED. This is something that would be extremely difficult to do if we were using **delay**()!

Task 3

In this task we are going to use a simple pushbutton switch to control an LED.

Firstly create the circuit shown in Figure 5, 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 pushbutton 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.

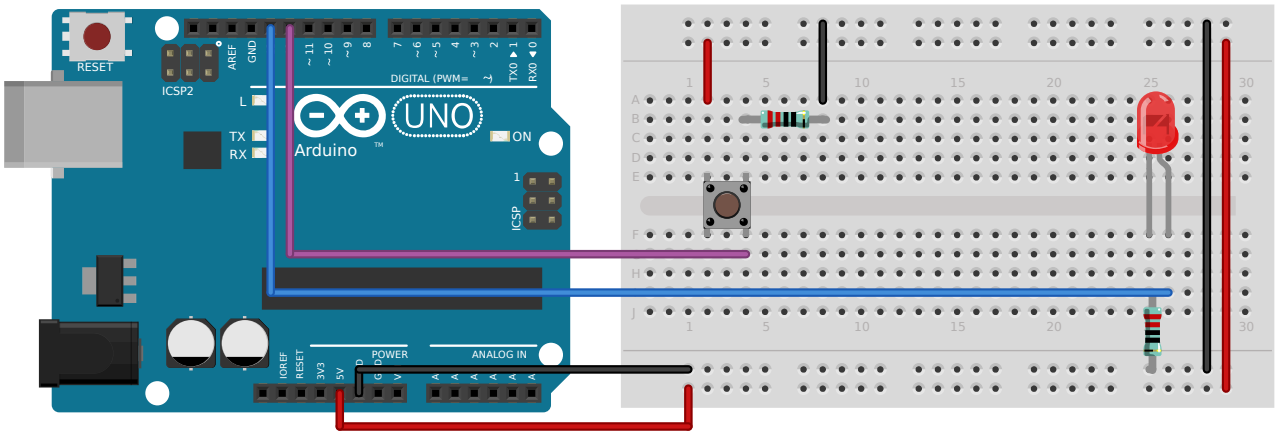

Figure 5 – The switch circuit

Now we wish to write some code to control this circuit. When the pushbutton is held down we want the LED to turn on, and then when the pushbutton is released we want the LED to turn off again. See code listing 3 for a simple program to accomplish this.

Code listing 3: LED on/off code

```
/*
 * led_on_off.ino
 *
 * simple sketch to turn an led on when the pushbutton is held
 * down and turn it off when it is released
 *
 */

// set pin numbers for button and led
const int button_pin = 12;
const int led_pin =  13;

// CODE

// set up code
void setup()
{
  // set pin i/o
  pinMode(led_pin, OUTPUT);
  pinMode(button_pin, INPUT);
}

// main program
void loop()
{
  // read the state of the pushbutton value:
  int button_state = digitalRead(button_pin);

  // check if the pushbutton is pressed.
  // if it is turn the led on
  if (button_state == HIGH)
  {
    digitalWrite(led_pin, HIGH);
  }
  // if it isn't turn led off
  else
  {
    digitalWrite(led_pin, LOW);
  }
}
```

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

s

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

1. 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).

2. 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. You should add “debouncing” to your program (which means checking the button state twice in a short period of time to ensure that the button was actually pressed and it wasn't just the contacts “bouncing”). Ask a member of staff is this is unclear.

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

Task 4

In this task we are going to use a potentiometer (commonly referred to as a pot) to control the blinking rate of an LED. To do this you should wire up the circuit shown in Figure 6.

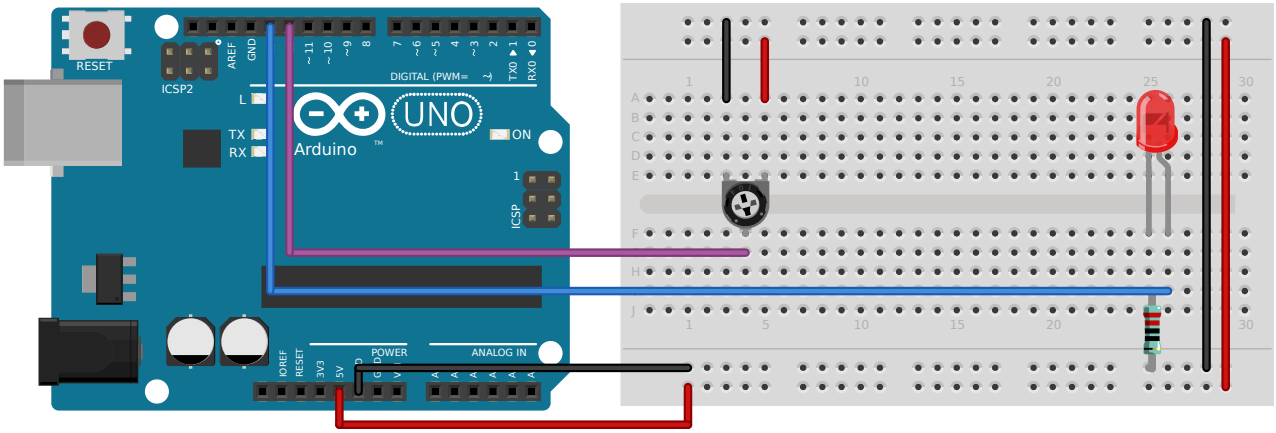

Figure 6 – The potentiometer circuit

This circuit has the two upper rails of the breadboard wired up 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 potentiometer then has one leg connected to +5V another leg connected to ground, and the centre leg (the "wiper") attached to analog input A0. We then connect the LED to digital pin 11.

We then need to write our program to control the LED (see code listing 4).

Code listing 4: Controlling the blink rate of an LED

```
/*
 * analog_blink_rate.ino
 *
 * simple program to adjust the blink rate of an led based
 * upon the value of a potentiometer
 */

// set pin numbers
const int led_pin = 11;
const int pot_pin = A0;

// initialise the state of the led
int led_state = LOW;

// CODE

// set up code
void setup()
{
  // set up the serial connection to enable debugging
  Serial.begin(9600);

  // set the pin i/o
  pinMode(led_pin, OUTPUT);
}

// main program
void loop()
{
  // get the value of the pot
  int pot_val = analogRead(pot_pin);

  // debugging
  Serial.print("Potentiometer value: ");
  Serial.println(pot_val, DEC);

  // flip the state of the LED
  if(led_state == HIGH)
  {
    led_state = LOW;
  }
  else
  {
    led_state = HIGH;
  }

  // then write it out and wait for the delay set by the pot
  digitalWrite(led_pin, led_state);
  delay(pot_val);

}
```

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

Note the use of *Serial.println()* for debugging in this sketch. That command lets us keep track of the value of our potentiometer by logging the analog value read from the potentiometer pin (pin A0) to the serial monitor. You will find that this functionality is extremely useful when writing programs of any complexity!

Q1 What is the maximum value of the potentiometer?

Q2 What is the minimum value of the potentiometer?

One common task when reading from analog sensors is calibrating those readings (i.e. tracking the highest and lowest values of a sensor during a start-up period and then using those values to remap the sensor range to be between another set of values).

Code listing 5 shows the process of tracking this sensor range during the start up period, and code listing 6 shows the remapping of this range to be a percentage.

Code listing 5:

```
// sensor pin
const int sensor_pin = A0;

// calibration variables
int sensor_value = 0;         // the current sensor value
int sensor_min = 1023;        // the minimum sensor value
int sensor_max = 0;           // the maximum sensor value

// CODE

// calibration routine (runs during setup() method)
void calibrate()
{
  // turn on led to signal the start of the calibration
  pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);

  // calibrate during the first ten seconds
  while (millis() < 10000)
  {
    sensor_value = analogRead(sensor_pin);

    // record the maximum sensor value
    if (sensor_value > sensor_max)
    {
      sensor_max = sensor_value;
    }

    // record the minimum sensor value
    if (sensor_value < sensor_min)
    {
      sensor_min = sensor_value;
    }
  }

  // signal the end of the calibration period
  digitalWrite(13, LOW);
}
```

Code listing 6:

```
void loop()
{
  // read the sensor
  sensor_value = analogRead(sensor_pin);

  // remap the sensor value using calibration information
  sensor_value = map(sensor_value, sensor_min, sensor_max, 0, 100);

  // if the sensor value is outside the calibrated range then constrain
  sensor_value = constrain(sensor_value, 0, 100);

  // debugging
  Serial.print("Potentiometer percentage: ");
  Serial.println(sensor_value, DEC);
}
```

Integrate and test this code. Try not turning the potentiometer through it's entire range during the calibration period.

Task 5

You are now required to use the integrated circuit temperature sensors supplied to measure the temperature of the outside world. The sensors we will be using are Analog Device TMP36s (see Figure 7) and extracts from the data sheet are provided in Appendix B. Ensure that you insert these temperature sensors the correct way round (see data sheet for correct wiring order).

They get very hot otherwise!!

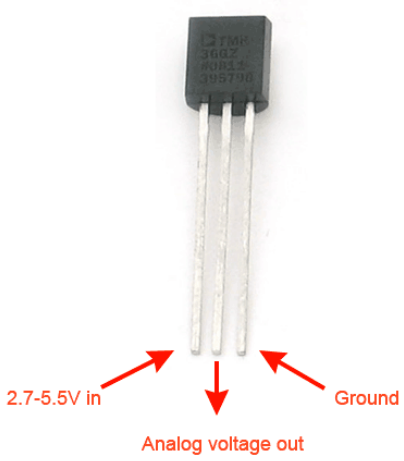

Figure 7 – A TMP36 temperature sensor

These sensors use a solid-state technique to determine the temperature. That is to say, they don't use mercury (like old thermometers), bimetalic strips (like in some home thermometers or stoves), nor do they use thermistors (temperature sensitive resistors). Instead, they use the fact as temperature increases, the voltage across a diode increases at a known rate. They amplify this voltage change by a precise amount to generate an analog signal that is directly proportional to the temperature.

Because these sensors have no moving parts, they are precise, never wear out, don't need calibration, work under many environmental conditions, and are consistent between sensors and readings. Moreover they are very inexpensive and quite easy to use.

As you can see from Figure 4 we get an analog voltage out (much like using a potentiometer). We will then have to convert that into a temperature using information from the data sheet (see Appendix B).

If you are unsure how to read the data sheet – ask a member of staff!!

Basic program

In this task, you are required to build and program a system that can output the current temperature (in degrees C) to the Arduino serial monitor window every 5 seconds. This system should also have a built in warning LED that lights up to indicate when the temperature exceeds a certain limit.

You should start with the program in code listing 2 as a basis and tweak it to achieve the above goals. When you have succeeded show your working system to a member of staff.

Additional functionality

Once you have this all working you should proceed to add additional features such as:

1. An indicator LED for switching the heating on when the temperature falls below a certain threshold.
2. A potentiometer that allow the heating set point to be adjusted (within a sensible range).
3. A potentiometer that allows the warning level set point to be adjusted (again, within a sensible range).
4. An integrated display (using a provided LCD screen – ask a member of staff!) showing key values such as current temperature and the current set points for heating and warnings. A guide for the LCD screens is available from blackboard.

Appendix A

We are now going to step through the blink.ino Arduino sketch entered above and explain the purpose of each element.

```
/*
 * blink.ino
 *
 * this is a simple Arduino program to blink an LED (the electronics
 * equivalent of hello world). This program turns an LED on for 1
 * second and then off for 1 second, and so on ...
 *
 * this program uses pin 13 for the LED
 */
```

This is a **block comment**. The text between the /* and */ symbols is ignored by the Arduino; however, it still serves an extremely important purpose – making the program readable to human eyes! This is important even if the code you write is never going to be seen by other people, as it helps you understand what a program does when you come back to it (you may think "oh that's simple, I don't need to explain what that does", but when you come back to it in a week / month / year it will suddenly not make any sense to you!).

At the start of each program you write you should put a block comment with the name of the program, a short description of what it does, and the author details (name, date, etc.).

```
int led_pin = 13;              // the LED is connected to pin 13
```

This is the first line of code that the Arduino will actually do anything with. It is a **statement** assigning a value to a **variable**. If you haven't done much programming before this statement may look unfamiliar to you. However, this is the general form of all Arduino statements (and the syntax is the same in many other programming languages). Firstly we need to tell the Arduino what the type of the variable is (in this case an integer), then the identifier for the variable (in this case ledPin), and finally what the value should be (in this case 13). All statements in the Arduino language should be ended by a semi-colon (;) as this lets the Arduino know that the statement is finished.

Note also that there is a single line comment after the statement (denoted this time by //). This is just to remind us what the statement does.

```
// this method runs once (when the sketch starts)
void setup()
{
  pinMode(led_pin, OUTPUT);    // sets the digital pin as output
}
```

This bunch of code is called a **method** (or procedure), and it is basically a collection of statements that we can then refer to by one name. We can see that this method doesn't return anything (it's return type is **void**) and doesn't require any inputs. The purpose of the **setup()** method is just to tell the Arduino what should be initialised (in this case one of the physical pins on the chip should be treated as an output).

```
// this methods loops continuously
void loop()
{
  digitalWrite(ledPin, HIGH); // turn LED on
  delay(1000);                // wait for a second
  digitalWrite(ledPin, LOW);  // turn LED off
  delay(1000);                // wait for a second
}
```

As with the **setup()** method we have just looked at, this **loop()** method also has no inputs or outputs. However, unlike the **setup()** method, this method contains multiple statements. This method calls two other methods – **digitalWrite()** and **delay()**. We will now look at these methods in a bit more detail:

digitalWrite() takes two inputs – a pin number (of type integer) and a logical value (either HIGH or LOW). In this sketch these values correspond to either turning the LED on (HIGH) or off (LOW). In more general terms this command switches the specified pin from 0 to 5V (HIGH) and back again (LOW)

delay() takes one input – the time (in milliseconds) to pause the execution of the method for.

It should also be noted that **loop()** - like **setup()** - is another special method. In this case **loop()** contains the main body of the program and, as the name suggests, is executed again and again until the Arduino is turned off (or another program is loaded on to it).

Appendix B

Extracts from the TMP36 data sheet

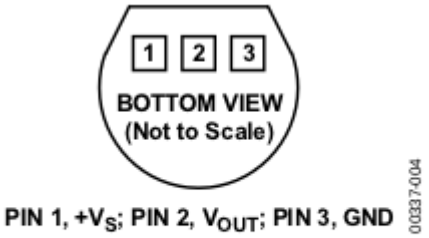

Figure 4. T-3 (TO-92)

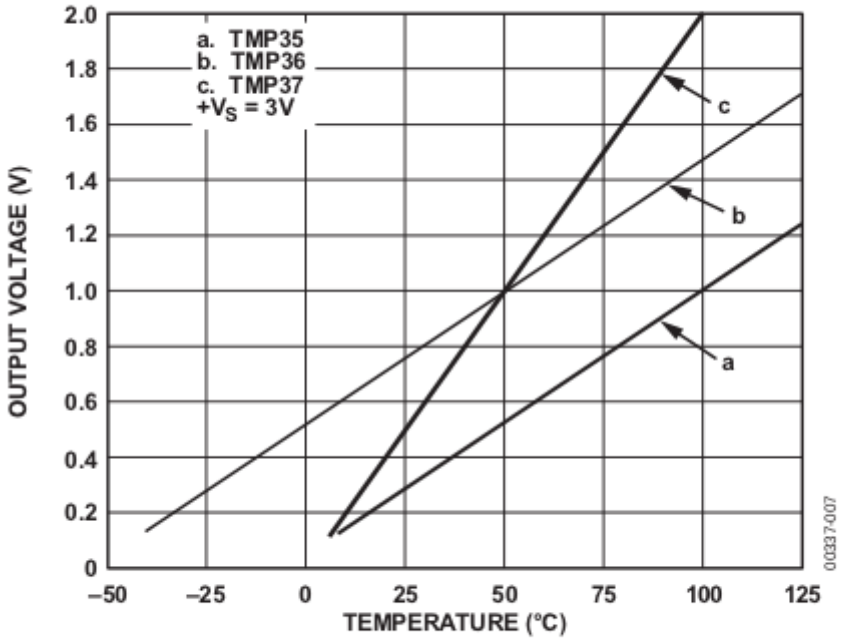

Figure 6. Output Voltage vs. Temperature

Table 4. TMP3x Output Characteristics

Sensor	Offset Voltage (V)	Output Voltage Scaling (mV/°C)	Output Voltage @ 25°C (mV)
TMP35	0	10	250
TMP36	0.5	10	750
TMP37	0	20	500

Check list

Task 1 – Program	
Task 2 – Program	
Task 2 – Exercises	
Task 3 – Program	
Task 3 – Additional functionality 1	
Task 3 – Additional functionality 2	
Task 3 – Additional functionality 3	
Task 4 – Program	
Task 4 – Q1	
Task 4 – Q2	
Task 4 – Calibration	
Task 5 – Basic program	
Task 5 – Additional functionality 1	
Task 5 – Additional functionality 2	
Task 5 – Additional functionality 3	
Task 5 – Additional functionality 4	

Feedback