Laboratory work: Arduino

Overview: The following handout has been created for a slightly longer laboratory session. Your aim in this lab is to complete the following tasks (traffic lights will take too long).

1. Write code and download it to the Arduino so as to make the integrated LED on pin 13 flash.

2. Write code to make an external LED toggle on/off using an external switch.

3. Write code to read the position of a potentiometer and send the value via the serial port to the terminal on the PC. Again if we had a bit more time we could send the data back to our own program.

4. Write code to display the potentiometer position on the LCD display.

Note once you program the Arduino it will restart and run when powered via the USB port or an external power supply. It does not need the PC to run.

Introduction to the Arduino

Overview: The Arduino is an example of a range of small microcontroller based embedded systems. The Arduino UNO used in this laboratory is built around the ATmega328 microcontroller. Microcontrollers consists of a single integrated circuit containing a cpu, memory, digital and analogue IO interfaces ("eye-Oh"- input/output).

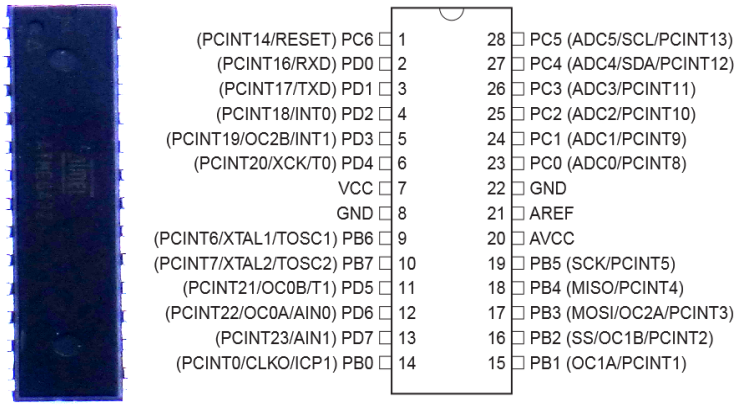
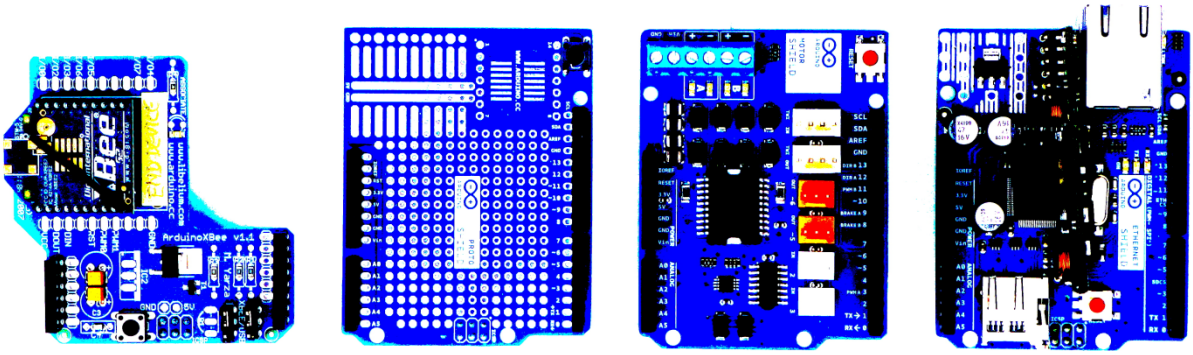


Figure 1: ATmega328 Microcontroller

In larger quantities these devices cost as little as €2. The Arduino is a platform that simplifies prototype development by placing a microcontroller on a circuit board that includes power supply, programming interface and discreet components used to set the clock rate and reset the controller on power up. In addition the Arduino has an excellent IDE (Integrated Development Environment) built around the "Processing" SDK. The Arduino can be programmed in C/C++ making it much easier to use than earlier systems built around assembly language. Due to its large following there is a huge database of code available to allow the Arduino to be interfaced to most devices without much effort. In addition to the large code base there are a large number of boards, known as "Shields", which can be plugged into the Arduino. Using shields simplifies hardware construction. Some typical shields are shown below.



Wireless communication Prototype Motor control LAN and SD card

Figure 2: Some Arduino shields

Once programmed, the Arduino can be run from a battery un-tethered from the computer. There are a number of different types of Arduino including Arduino Mega (more speed, interface pins and memory), Arduino Nano (small physical size) and the Arduino ADK (Android interface). The Arduino used in this lab is the Arduino UNO, shown below. There are 13 digital IO pins which can be used to detect (or set) a logic level. Bit 0 and 1 are used for serial communication so take care if you decide to use these.

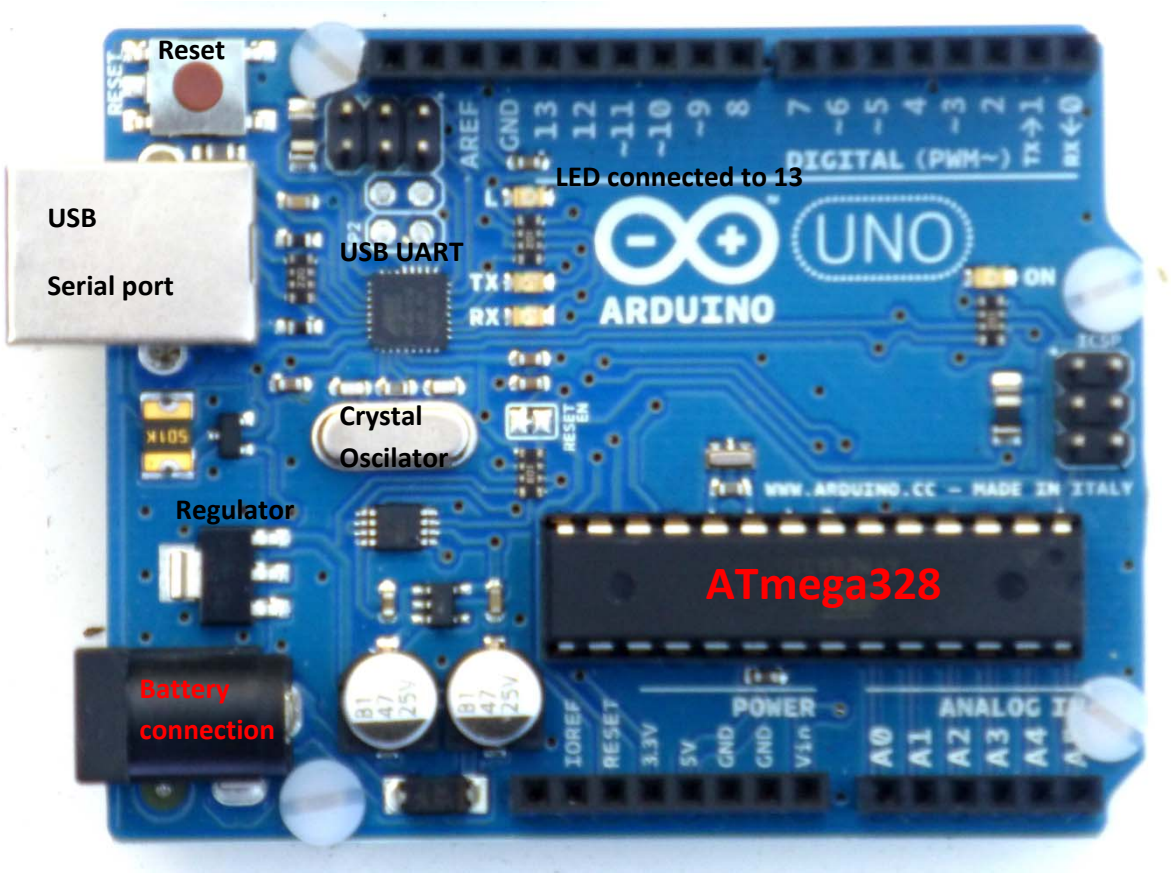


Figure 3: Arduino UNO

To simplify laboratory work, we have created an Arduino with a prototype board, with a breadboard, speaker and LCD display. This simplifies construction but means that some of the digital IO pins have been allocated. The pins in use are as follows,

LCD Display: Reset-pin7, Enable-pin10, Data4-pin8, Data5-pin11, Data6-pin9, Data7-pin12

Loudspeaker: pin6

Serial communication: RX (Receive)-pin0, TX (Transmit)-pin1

So the board shown in figure has analogue input pins A0 to A5 and digital I/O pins 2 3 4 5 13 available for general use. You may of course unplug the proto typing board or LCD connector to gain more pins.

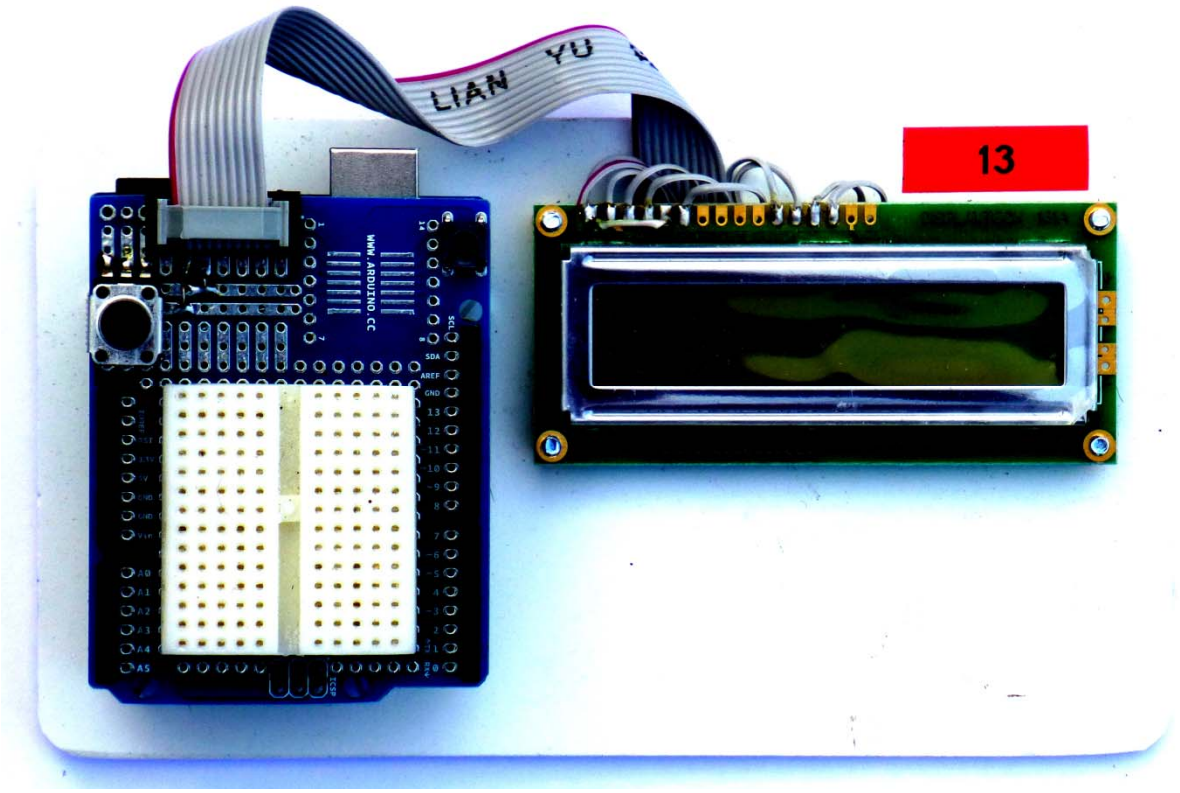


Figure 4: Arduino UNO with prototype board, speaker and LCD display.

Blink LED: To program the device launch the Arduino SDK and choose *File-Examples-Basics-Blink* and then press the arrow button to upload the program to the Arduino. You will probably need to select the correct communication port if it throws an error when downloading, *Tools-Serial Port-* and choose another port on offer.

```
int led = 13;

void setup()
{
  pinMode(led, OUTPUT);
}

void loop()
{
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

The Blink program contains two methods (functions), *setup()* and *loop()*. *Setup()* runs once following download or following a reset or power up. *Loop()* is effectively an in infinite loop (e.g. *while(true){...}*) into which you can place code that you wish to run repeatedly. The *pinMode()* allows you to set the data direction on an I/O port, in this case pin13 is configured as an output. *DigitalWrite()* then allows you set the state of the pin as either logic low (0 Volts) or logic high (5 Volts). The *delay()* functions provides a delay measures in

milliseconds. The program configures pin 13 to be an output and then repeatedly switches the pin from 5 volts for one second to 0 volts for one second. An orange surface mounting LED is connected via a resistor to pin 13 on the Arduino. You should see this flashing, well done!

Button press and LED

The following circuit is required to interface a switch to the input of a microcontroller. It consists of a "pull up resistor" with a value of 10,000Ohms (or 10kΩ) connected to 5V. The other end is connected to the digital input of the microcontroller (pin 2) forcing the input to logic *high*. A switch can then be used to connect pin 2 to ground forcing the input to 0 volts or logic *low*. The resistor limits the current.

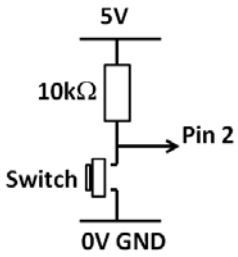


Figure 5: Interface circuit to connect switch to input logic.

The following circuit is required to connect an LED to the output pin of a microcontroller. An LED is a current controlled device, however the microcontroller use potential (voltage) to define the output. The maximum current that can flow through and LED is typically 20mA (0.02Amps), the voltage drop across a red led is typical 1.2V. Using Ohms law we can calculate the resistor required to limit the current through the led as follows.

$$V = IR$$
$$5 - 1.2 = 0.02R$$
$$R = 190\Omega$$

This would give us maximum permissible brightness, for this lab we will use a 1kΩ resistor, bright enough!

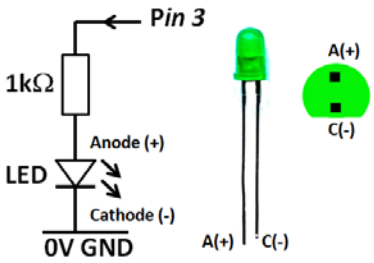


Figure 6: Interface circuit to connect a led to output logic (current sourcing).

Note: on many microcontrollers the outputs can only sink current rather than source it (as above). The Arduino can both sink and source current. When using these devices you will need to connect the anode to the 5V supply, the cathode is then connected to logic output via a limiting resistor. The main change in this case is that logic high would turn the LED off and logic low would turn the LED on. This alternative wiring is shown below and can be used with the Arduino.

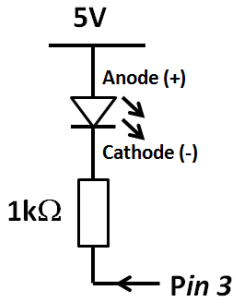


Figure 7: Interface circuit to connect a led to output logic (current sinking).

The following LED and button circuit can be wired on the bread board, a suggestion is shown in figure 8. The pins on the breadboard are connected in groups of five running from the side of the board and terminating in the centre.

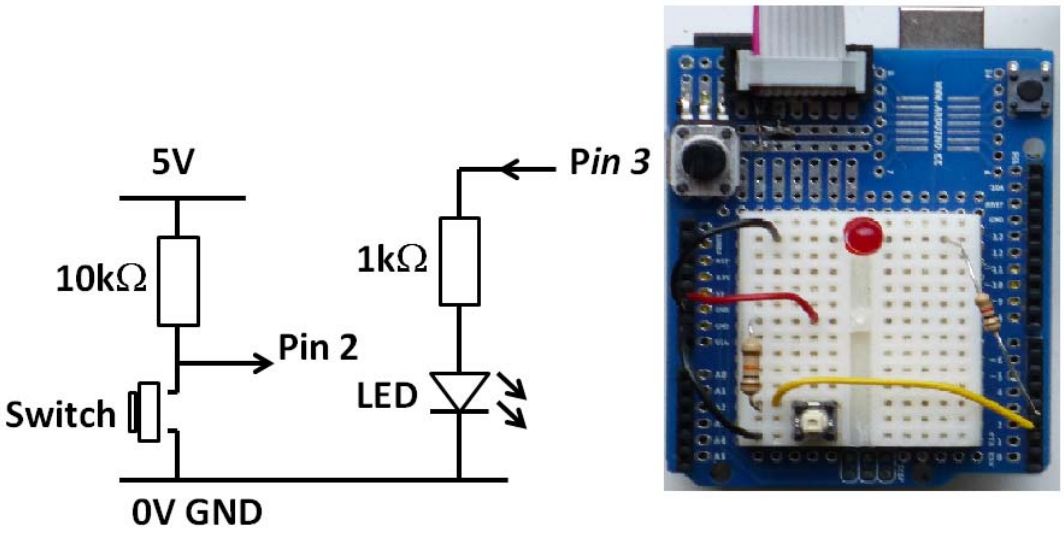


Figure 8: Button and LED interface circuit built on bread board

The button leads may need to be formed to fit in a 3x3 grid, the connections are shown below.

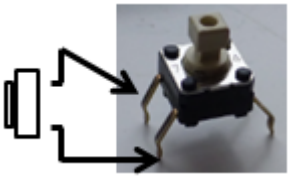


Figure 9: Button connections

Build the circuit and enter the following code into a new Arduino project. Run the program and you should see the LED turn on each time you press the button.

```
void setup()
{
  pinMode(2, INPUT);
  pinMode(3, OUTPUT);
}

void loop()
{
  if(digitalRead(2)==LOW)
  {
   digitalWrite(3, HIGH);
  }
  else
  {
    digitalWrite(3, LOW);
  }
}
```

Toggle action: The following code detects a falling edge on pin 2. This falling edge is then used to toggle the action of the light. Pressing the button turns the led on and pressing it again turn it off. "Contact bounce" can create a sequence of on/off signals each time the button is pressed and released as the contacts bounce before making a final make or break circuit. The solution in this example is to use a small delay after the edge detector.

```
int newState=HIGH,oldState=HIGH,ledState=LOW;

void setup()
{
  pinMode(3, OUTPUT);
  pinMode(2, INPUT);
}

void loop()
{
 newState = digitalRead(2);

 if ((oldState == HIGH)&&(newState==LOW))
 {
  delay(20);

  if(ledState==HIGH)
  {
    digitalWrite(3, LOW);
   ledState=LOW;
  }
  else
  {
    digitalWrite(3, HIGH);
    ledState=HIGH;
  }
 }

 oldState=newState;
}
```

Traffic lights: To build traffic light controller is a rite of passage, it goes back to the foundation of microprocessors themselves as this was one of their first applications. The following circuit and suggestion may be of use to get you started. You could consider adding a pedestrian light (red and green) as well as the traffic light (red, amber and green) to your design. The definitive problem is to control a cross roads with single pedestrian crossing; you would need to unplug the LCD display to have enough pins to do this and this is perhaps for another day.

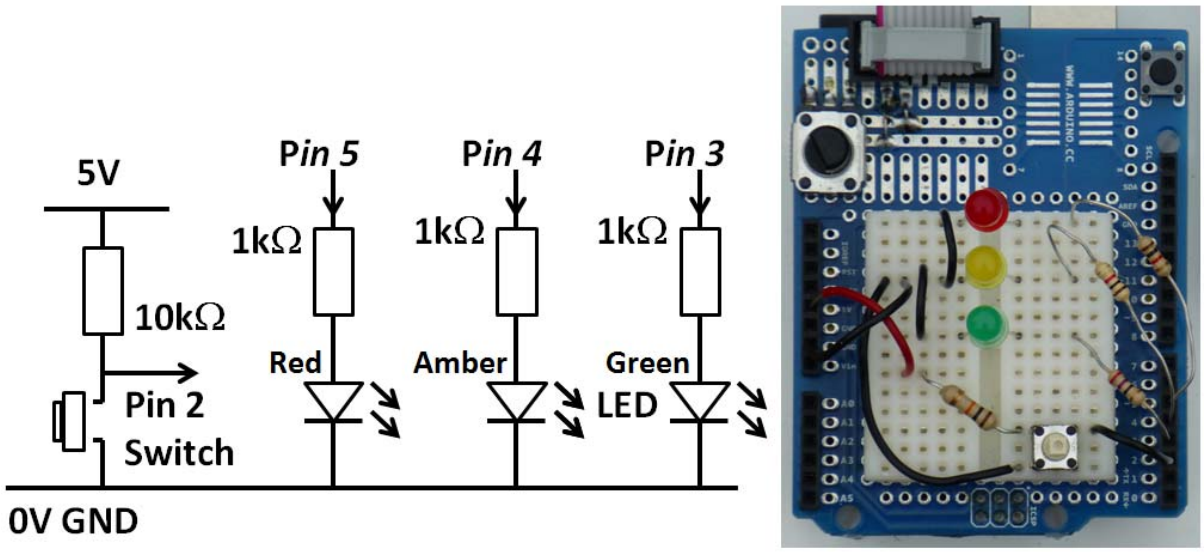


Figure 10: Traffic light sequencer circuit.

Three colour LED: A special LED containing separate red, green and blue LEDs is available. The connection to the LED is as follows, there are two blue LEDs in the device as they can be dimmer. See if you can create an program that cycles through each of the following sates (off, red, green, blue, magenta, cyan, yellow, white), waiting a half a second on each colour.

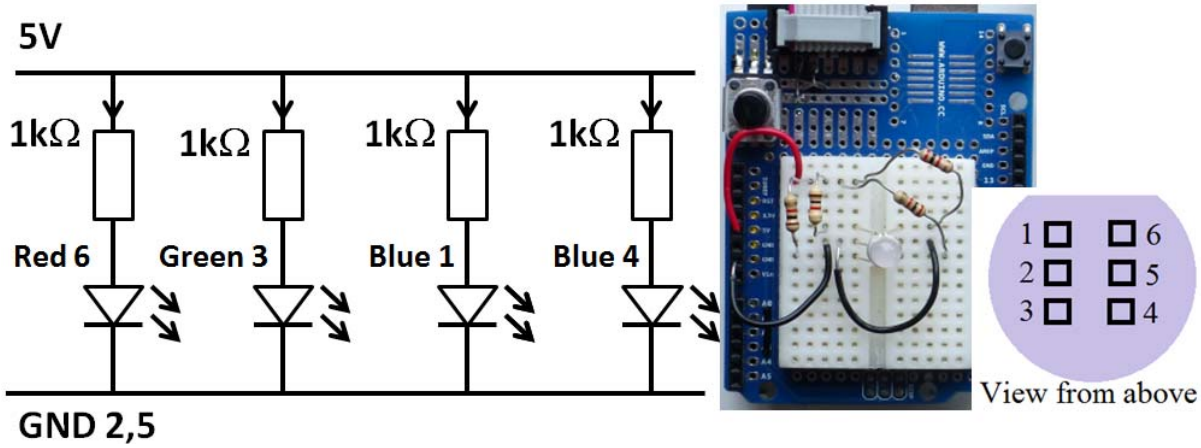


Figure 11: Lamp test circuit for three colour LED.

Analogue to digital conversion: Many devices produce an analogue signal. An analogue signal is continuous and could include examples such as speed, light intensity, sound level or temperature. The Arduino can convert an analogue into a digital binary number representing the signal. The process is known as analogue to digital conversion. The digital value representing the analogue signal is limited to a discrete range of integer values. The ADC in the Arduino has a 10 bit precision so produces a value between 0 and 2^{10}-1=1023 for an input rage of 0 Volts to 5 Volts. A potentiometer is a device with three connections. Two connections are made to either end of a resistor, this is known as the track, the third is connected to a wiper that can make moving connection anywhere along the length of the track. If either end of the track is connected to a voltage source then the wiper can be used to produce a potential (voltage) anywhere between 0 and the battery potential.

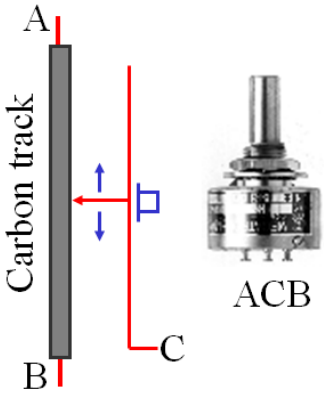


Figure 11: The three connections to a potentiometer.

Connect the wiper of the potentiometer (yellow wire) to pin A0, and the black wire of the track to 0V and the red wire to 5V and run the following program. The program reads the value on the connected to A0 and sends it via the serial communication port to the PC.

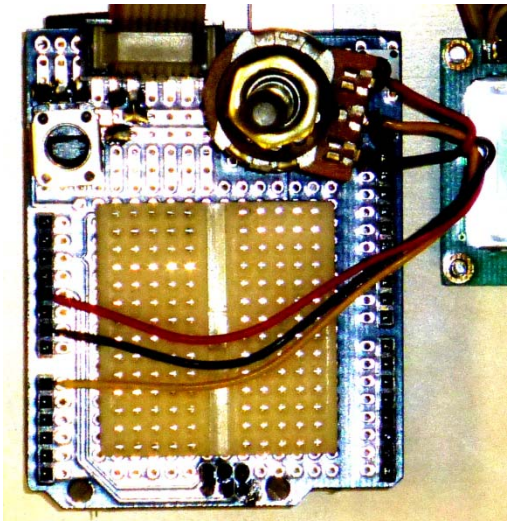


Figure 12: Potentiometer connected to Arduino, A0-Yellow, 5V-Red, GND-Black.

Click on the Serial Monitor button, 

, of the Arduino IDE. Adjust the position of the wiper and note the range of values produced over the range [0,1023].

```
int sensorValue;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(10);
}
```

The potentiometer has rotation range of 0 to 270 degrees. Arduino provides a means of scaling (or calibrating) efficiently as follows. You could also use floating point calculation to achieve the same result.

```
int sensorValue;
int wiperPosition;
float x;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  sensorValue = analogRead(A0);
  wiperPosition = map(sensorValue, 0, 1023, 0, 270);
  Serial.println(sensorValue);
  delay(10);
}
```

Digital to analogue conversion: Many microcontrollers contain a device to convert a digital value back into an analogue signal (DAC, digital to analogue convertor), this is how most sound cards generate their audio signal. The Arduino has a limited DAC that produce a PWM (Pulse width modulated signal) based on the value sent to the digital pin. The duty cycle of the square wave produce by the pin is adjusted by the DAC value, (e.g. 255 always on, 0 always off and 128 on/off 50% of the time). Only digital pins with a tilde ("~") have this functionality. The DAC value on the Arduino is in the range 0 to 255 (eight bit). The following code allows you to adjust the brightness of an LED (keep the 1K series resistor connected to ground to limit current) using the potentiometer. The LED should be connected to digital connection pin 3, similar to the LED connection in figure 8.

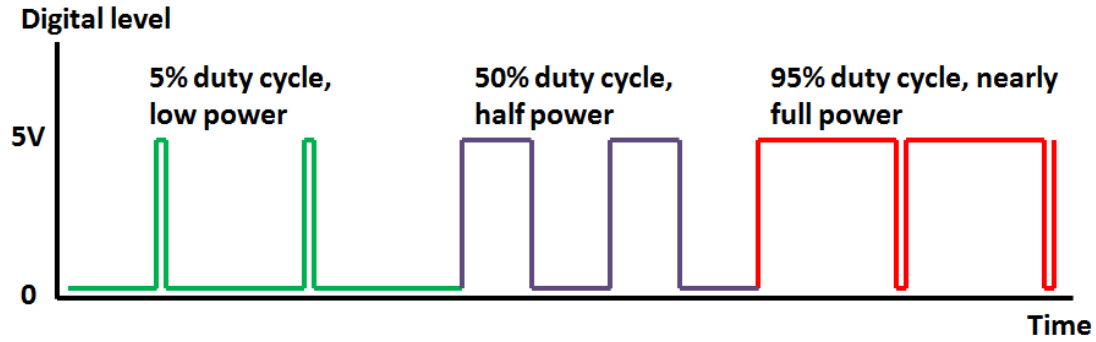


Figure 12: Digital to analogue conversion using adjustable duty cycle square wave.

```
int ADC_value;
int DAC_value;

void setup()
{
}
void loop()
{
  ADC_value = analogRead(A0);
  DAC_value = map(ADC_value, 0, 1023, 0, 255);
  analogWrite(3, DAC_value);
  delay(10);
}
```

There are many devices that you can connect to the digital and analogue connections of the Arduino. See figure 13 for some suggestions. Motors, speakers and solenoids may require the use of additional shields or FET or transistor to increase the current required to power the device.

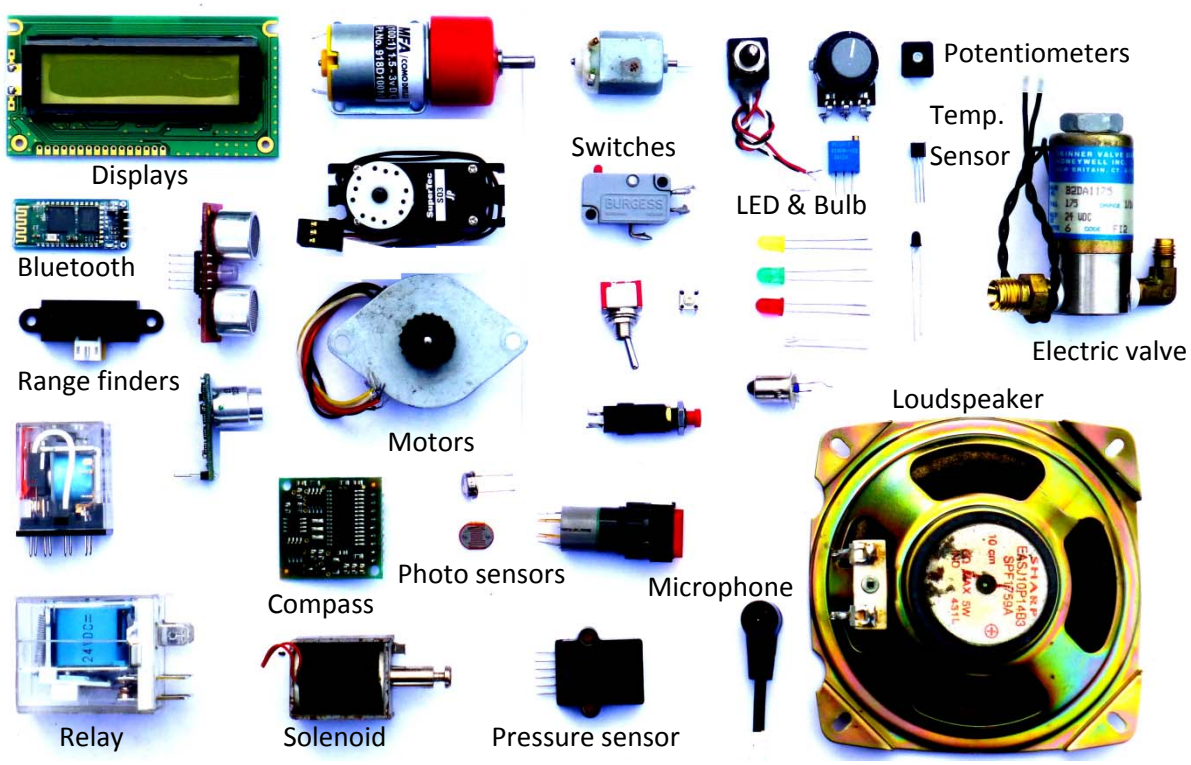


Figure 13: Some devices straightforward to connect to an Arduino

LCD display: Instead of sending values to the serial port it is straightforward to connect an LCD display to the Arduino and display the values on it. The LCD has 6 digital connections to communicate with the Arduino. In addition an adjustable voltage is required to set the screen contrast (achieved using a 10k potentiometer) and 5V is required to power the device. To simplify the connections a connector has been added to the prototype board. The LCD display used splits the16x1 character display into a 8x2 character display. See the "Hello world" program in the example folder if you wish to access all 16 characters. For now, connect the LCD display and run the following program adjusting the potentiometer show change the value shown on the LCD display. Remember to adjust the contrast control if you see nothing at run time.

```
#include <LiquidCrystal.h>
int sensorValue;
int wiperPosition;
LiquidCrystal lcd(7, 10, 8, 11, 9, 12); // Reset,Enable,D4,D5,D6,D7

void setup()
{
  lcd.begin(16, 2);
}

void loop()
{
  lcd.setCursor(0, 0);
  sensorValue = analogRead(A0);
  wiperPosition = map(sensorValue, 0, 1023, 0, 270);
  lcd.print("Pos: ");
  lcd.print(wiperPosition);
  lcd.print("  ");
  delay(10);
}
```

Arduino to PC: The Arduino and similar devices provide an effective way of interfacing programs and application to external hardware. The Arduino contains a serial communication UART (Universal Asynchronous Receiver and Transmitter) within its USB interface. This appears to the programmer of the PC as a serial communication port. The following program reads two of the ADC ports and one of the digital ports and prints the values in ASCII format to a string sent to the serial communication port. Using C# (or Java or cpp or processing etc) we can read data from the serial port. Run the following program on the Arduino and check if it is sending data in CSV (comma separated value format) using the IDE Serial monitor.

```
void setup()
{
  Serial.begin(9600);
  pinMode(2, INPUT);
}

void loop()
{
  Serial.print(analogRead(A0));
  Serial.print(",");
  Serial.print(analogRead(A1));
  Serial.print(",");
  Serial.println(digitalRead(2));
  delay(20);
}
```

Launch Visual studio 2010 and create a new WindowsForm application called “Arduino”, if you chose a different name then you will need to change the “namespace” references to your new name. Using the toolbox add a single *label*, *timer* and single *serialPort* to the form. Right click on the serialPort and press the lightning bolt button and choose to add a *serialPort1_DataRecieved* event manager to the designer (you can see the code you have added by looking at the file Form1.Designer.cs). In the properties for the timer, enable the timer and set the delay to 50mS (20 per second). Replace the existing Form1.cs code with the following (using cut and paste), change the *COM* port value to that used on your machine and run the code. Each time the PC receives data from the Arduino the *serialPort1_DataRecieved method()* is called. This method parses the string received to extract the Analog0, Analog1 and button states. The timer is used to repeatedly display these values using a label. Games controllers work in a similar way, typically using a HID (Human Interaction Device, a bit more fiddly but not impossible) interface rather than serial communication port but essentially the same ideas.

Note: You may notice that A1 follows A0 even though only A0 is connected to a potentiometer. There is only one ADC in the Arduino and a digital switch to select which input to read. To reduce the cross talk you could connect A1 to GND forcing it to zero or you could connect it to a second potentiometer if it is available.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.IO.Ports;
using System.Text.RegularExpressions;

namespace Arduino
{
    public partial class Form1 : Form
    {
        String Instring;
        int Analog0 = 0;
        int Analog1 = 0;
        int Digital2 = 0;

        public Form1()
        {
            InitializeComponent();
            serialPort1.PortName = "COM8";  // May need to change
            serialPort1.BaudRate = 9600;
            serialPort1.Parity = System.IO.Ports.Parity.None;
            serialPort1.DataBits = 8;
            serialPort1.StopBits = System.IO.Ports.StopBits.Two;
            serialPort1.Handshake = System.IO.Ports.Handshake.None;
            serialPort1.Open();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        }

        private void serialPort1_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            lock(this)
            {
            Match position;
            Instring += serialPort1.ReadExisting();
            if (Instring.Length > 20)
            {
                position = Regex.Match(Instring, @"\n[0123456789,]*\r");

                if (position.Success)
                {
                    String data = Instring.Substring(position.Index + 1, position.Length - 2);
                    String[] sdata = data.Split(',');
                    if (sdata.Length == 3)
                    {
                        Analog0 = Convert.ToInt32(sdata[0]);
                        Analog1 = Convert.ToInt32(sdata[1]);
                        Digital2 = Convert.ToInt32(sdata[2]);
                        Instring = "";
                    }
                }
            }
            }
        }


       private void timer1_Tick(object sender, EventArgs e)
        {
            label1.Text = Analog0.ToString() + " " + Analog1.ToString() + " " + Digital2.ToString();
        }
    }
```