Sheffield Hallam University
School of Engineering and Built Environment

Activity ID	Activity Title	Laboratory Room No.	Level
Lab 3	Analog output with the Arduino	4302	Beginner

Equipment (per student/group)

Number	Item
1	PC
1	Arduino kit

Learning Outcomes

	Learning Outcome
3	Develop good laboratory practice and demonstrate knowledge, understanding and ability to safely use relevant materials, equipment, tools, processes or products.
4	Demonstrate competent use of prototyping, modelling and basic analytical techniques.

Analog output with the 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. The heart of this prototyping system is the Arduino Uno microcontroller board itself which is based on the commonly used ATMega328 chip.

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 – **output**.

By the end of this set of laboratory exercises you will understand how to using pulse width modulation to simulate an analog output with the Arduino microcontroller.

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)!

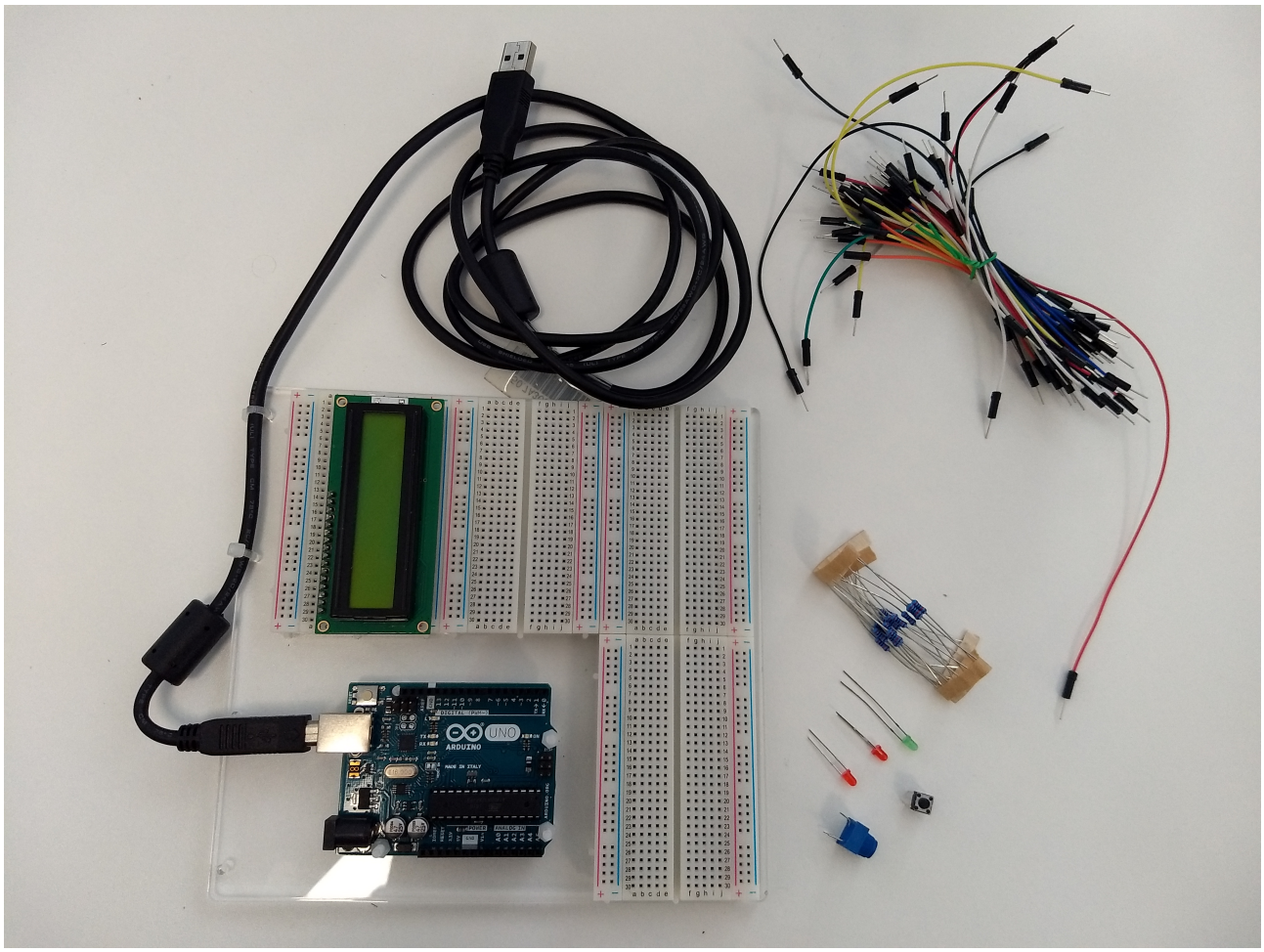


Figure 1 – The necessary equipment for this lab (don't worry if you have LEDs of a different colour or slightly different breadboards)

Task 1

This time we are going to re-use the same circuit as in task 3 in lab 2 (see Figure 4 in lab 2) – except we are going to wire the LED up to pin 11 (rather than 13). This is because the ability to do pulse width modulation is restricted to certain pins. As we did last in the last lab, we will use the analogRead function again to read the value of the potentiometer, but this time we will use pulse width modulation to perform an analogWrite. See code listing 1 for the complete program.

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave (i.e. a signal switched between on and off). This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In Figure 2 below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduinos PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time).

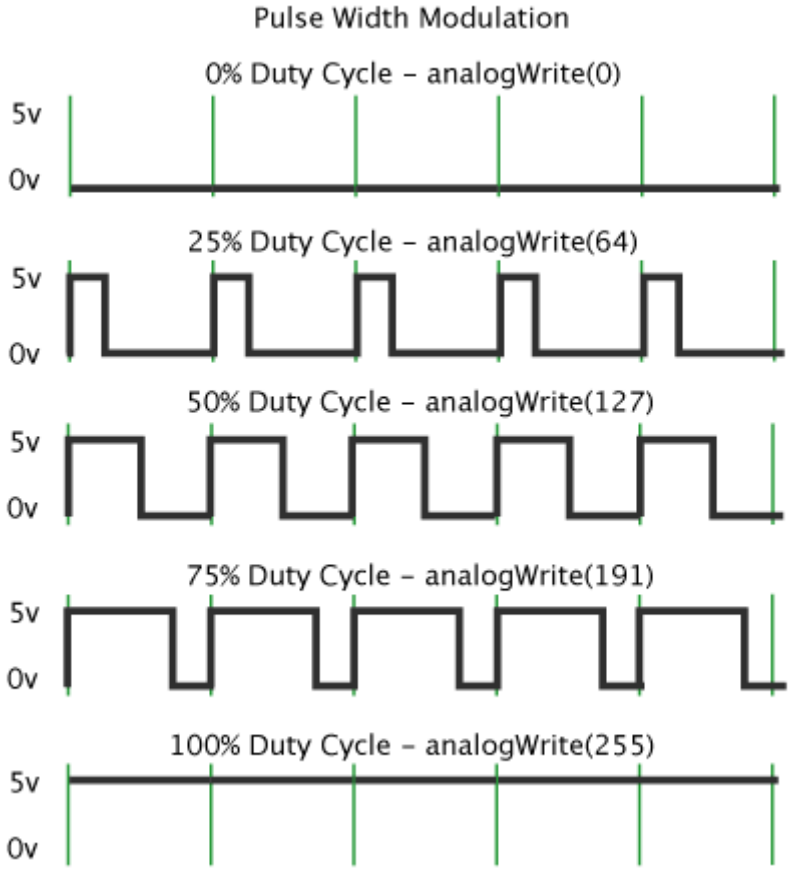


Figure 2 – Illustration of pulse width modulation[1]

1 Taken from http://www.arduino.cc/en/Tutorial/PWM

PWM is only available on certain output pins of the Arduino (11, 10, 9, 6, 5, and 3 on the Arduino Unos we are using in class).

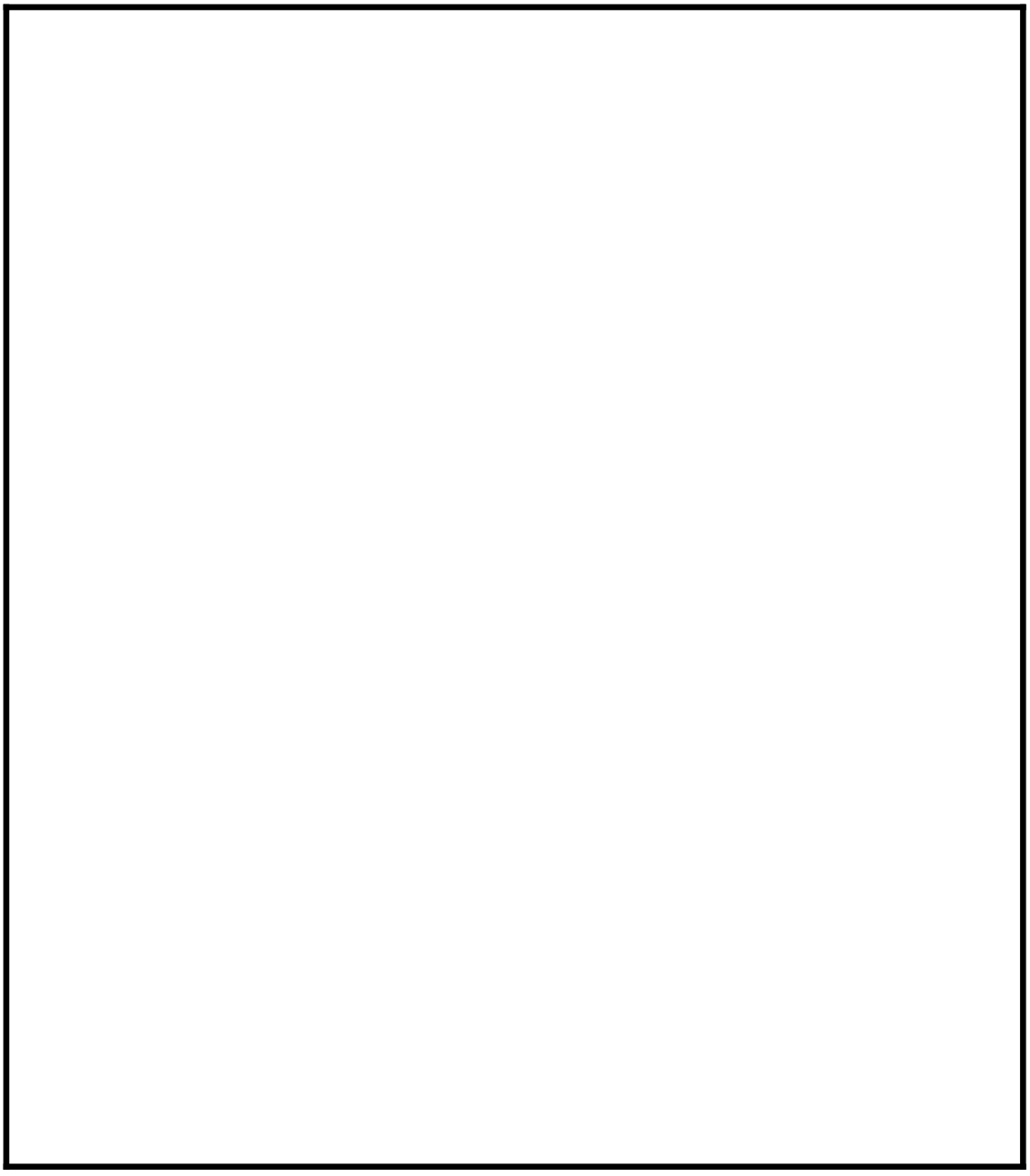
Code listing 1: Analog LED fade

```
/*
 * 1_analog_led_fade.ino
 *
 * simple sketch to fade an led based on the potentiometer input
 *
 * the led is on pin 11 because pin 11 is capable of pwm (it has a ~ symbol next
 * to it)
 *
 * author: prof. alex shenfield
 * date:   2024/11/12
 */

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

// initialise the brightness value
int brightness = 128;

// CODE

// set up the initial state of the program
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 brightness (this value needs to be between 0 and 255, so we need
  // to divide the pot value by 4 - as the analogRead of the pot is in the
  // range 0 to 1023)
  int val = analogRead(pot_pin);
  brightness = val / 4;

  // note: we could also have used the map() function above

  // debugging
  Serial.print("Potentiometer value: ");
  Serial.println(val, DEC);
  Serial.print("Brightness value: ");
  Serial.println(brightness, DEC);

  // set the brightness of the led
  analogWrite(led_pin, brightness);

  // wait slightly to allow dimming effect to be seen
  delay(30);
}
```

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

Task 2

Now you are going to use everything you have learnt in this and the previous lab exercises to build and program a complete circuit to crossfade a set of different colour LEDs based on the analog input from a potentiometer. Sketch your circuit below, making sure that you specify the pins that the potentiometer and the LEDs are attached to:



Once this circuit has been built you need to program it. However, before you write any code you should sketch out the algorithm. This should be a set of unambiguous steps that the Arduino can use to control the circuit. Below is a rough outline of my version of this algorithm – work through this algorithm and make sure you understand it.

```
set up i/o

read potentiometer

if potentiometer value is in the bottom half of its range (i.e. potentiometer < 512)

    normalise the potentiometer value to be in the range 0-255

    led 1 = 255 – normalised potentiometer value
    led 2 = normalised potentiometer value
    led 3 = off

else if the potentiometer value is in the top half of its range (i.e. potentiometer ≥ 512)

    normalise the potentiometer value to be in the range 0-255

    led 1 = off
    led 2 = 255 - normalised potentiometer value
    led 3 = normalised potentiometer value

end
```

Graphically, the output of the LEDs looks something like Figure 3 (below).

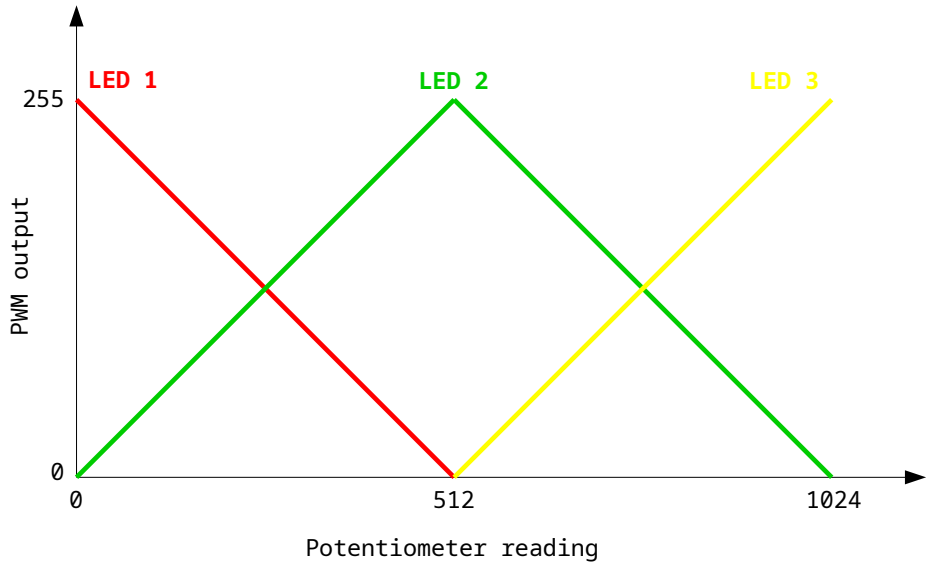


Figure 3 – Graphical representation of LED outputs

We can normalise the potentiometer value by using the Arduino map function:

https://www.arduino.cc/en/Reference/Map

In the lower half of the potentiometer range (i.e. potentiometer value < 512) we can use the code:

```
output_val = map(input_val, 0, 511, 0, 255);
```

Q1 What is the code for remapping the potentiometer input value in the top half of the potentiometer range (i.e. potentiometer value ≥ 512)?

Q2 What are the values of the LEDs in the following cases?

Potentiometer value	**LED values**
0	
256	
512	
768	
1023	

Only now should you attempt to write your program! Note: if you are struggling with translating your algorithm into code, there is a sample solution provided with the other lab code.

Check list

Task 1 – Program	
Task 2 – Circuit	
Task 2 – Algorithm	
Task 2 – Q1	
Task 2 – Q2	
Task 2 – Program	

Feedback/ reflections