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

Activity ID		Activity Title			Laboratory Room No.	Level
Lab 103		Driving outputs 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

Driving outputs 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. We then looked at some of types of sensors we can use with the Arduino system to give an embedded system the ability to "see" the world. However, it's all well and good reading data from the outside world, but to be useful an embedded system also has to be able to process that data and **react to it**.

In this laboratory session we are going to deal with an extremely important aspect of any kind of embedded system – **output**. In the first laboratory session we focussed on simple digital output using LEDs; however, many real-world situations require an analog output to control a system (such as motors, speakers, and communication systems). This can be a problem when using digital systems such as a microcontroller!

By the end of this set of laboratory exercises you will understand the concept of pulse-width modulation – a way of providing a (sort of) analog output using a digital signal – and how to use it with the Arduino MKR WiFi 1010.

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)!

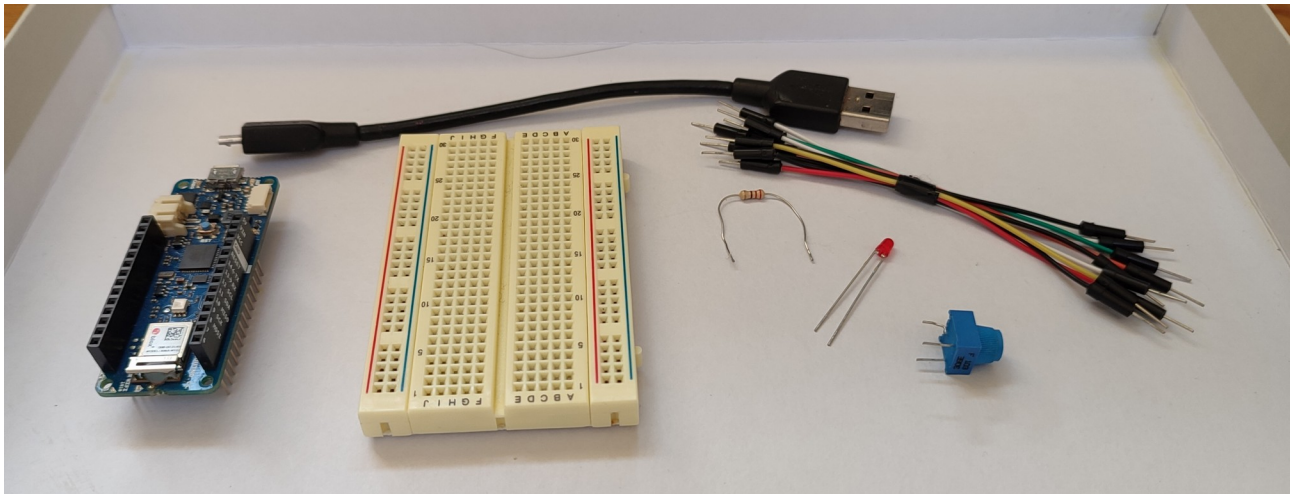

Figure 1 – The necessary equipment for this lab (don't worry if you have LEDs of a different colour or slightly different breadboards)

Task 1

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 the Arduino's 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).

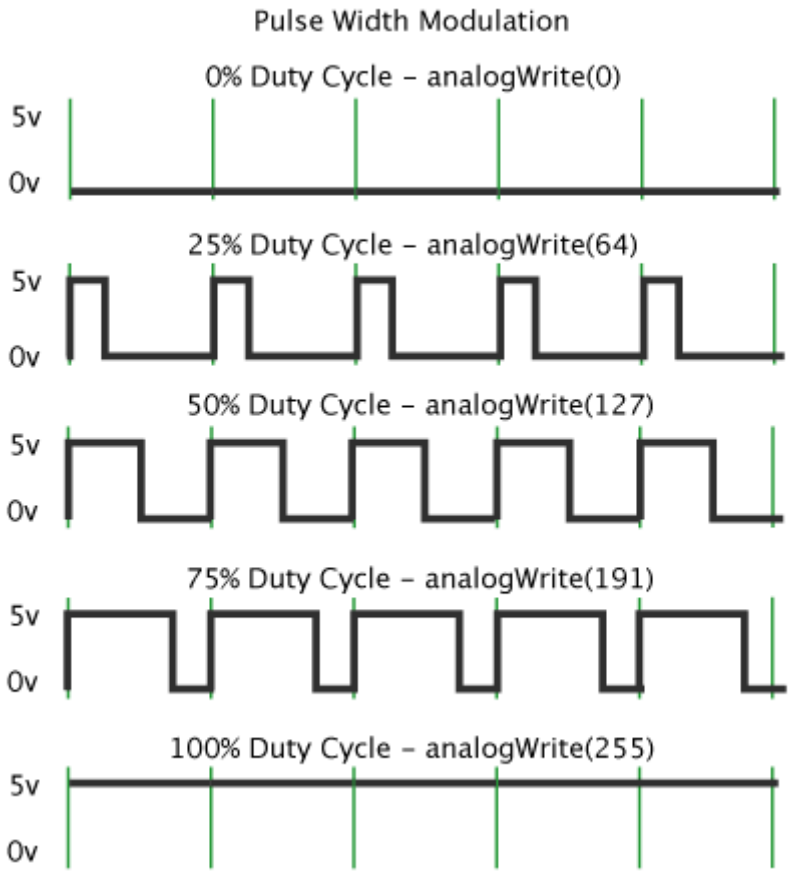

Figure 2 – Illustration of pulse width modulation[1]

PWM is only available on certain output pins of the Arduino MKR WiFi 1010 (2, 3, 4, and 5) and these are denoted by a "~" on the pin label.

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

In this task we will use the analogRead function (as we did in the LDR example last week) to read the value of a potentiometer and then use pulse width modulation to dim an LED.

Firstly create the circuit shown in Figure 3, below. This wires up the two upper rails of the breadboard to the +3.3V and ground pins on the Arduino board (+3.3V to the lower rail and ground to the top rail) allowing us to easily access the +3.3V supply and ground. We then connect one leg of the potentiometer to +3.3V, one leg to ground, and then the middle leg to pin A0 (which is one of our analog inputs). Finally, we wire up an LED to pin 5 (one of the PWM capable pins).

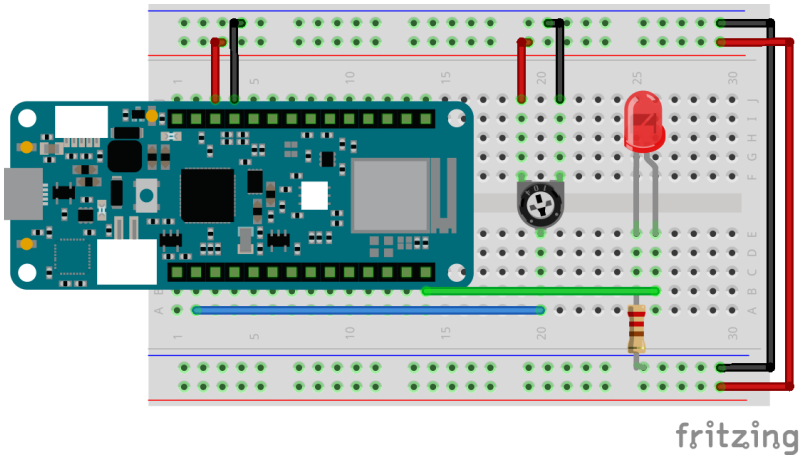

Figure 3 – LED dimmer circuit

See code listing 1 for the complete program.

Code listing 1: Code to dim an LED based on the value of a potentiometer

```
/*
* 1_led_dimmer.ino
*
* this is a simple Arduino program to read from a potentiometer and use
* that reading to dim an led using pwm
*
* this program uses pin A0 for the analog sensor and pin 5 for the led
*
* author: alex shenfield
* date:   01-09-2022
*/

// INITIAL VARIABLES

// the pot is connected to A0 and the led is connected to pin 5
const int pot_pin = A0;
const int led_pin = 5;

// CODE

// 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()
{
  // read the potentiometer (varies between 0-1023)
  int pot_value = analogRead(pot_pin);

  // remap the potentiometer value to between 0 and 255 (as that is
  // what the pwm signal needs) and send the pwm signal to the led
  int pwm_value = map(pot_value, 0, 1023, 0, 255);
  analogWrite(led_pin, pwm_value);
}
```

Enter this program, upload it to the board, and make sure everything works.

Task 2

In this task we are going to use PWM to control the built-in RGB LED on the Arduino MKR WiFi 1010 board.

RGB (Red, Green, Blue) LEDs usually look pretty much like any other LED – however, inside the package are actually three separate LEDs (one red, one green, and one blue) that can be controlled individually. By varying the brightness of each of these we can make any colour that we want!

Figure 4 shows the location of the RGB LED on the MKR 1010 – just to the left of the NINA-W102 WiFi module.

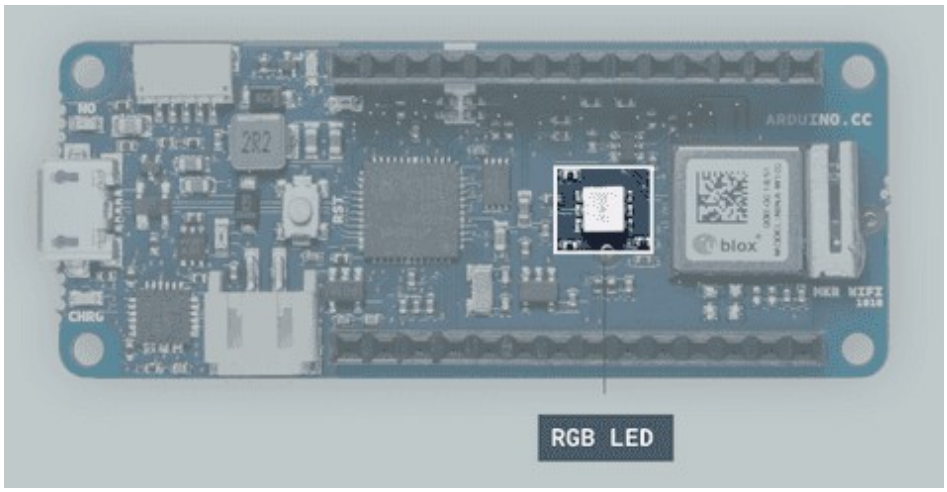

Figure 4 – The built-in RGB LED on the Arduino MKR WiFi 1010

To control this built-in LED, we'll actually have to use the WiFiNINA library[2] (as well as using PWM to change the colours). Figure 5 shows the installation of this library via the Arduino library manager.

2 For stand-alone RGB LEDs you won't have to do this!

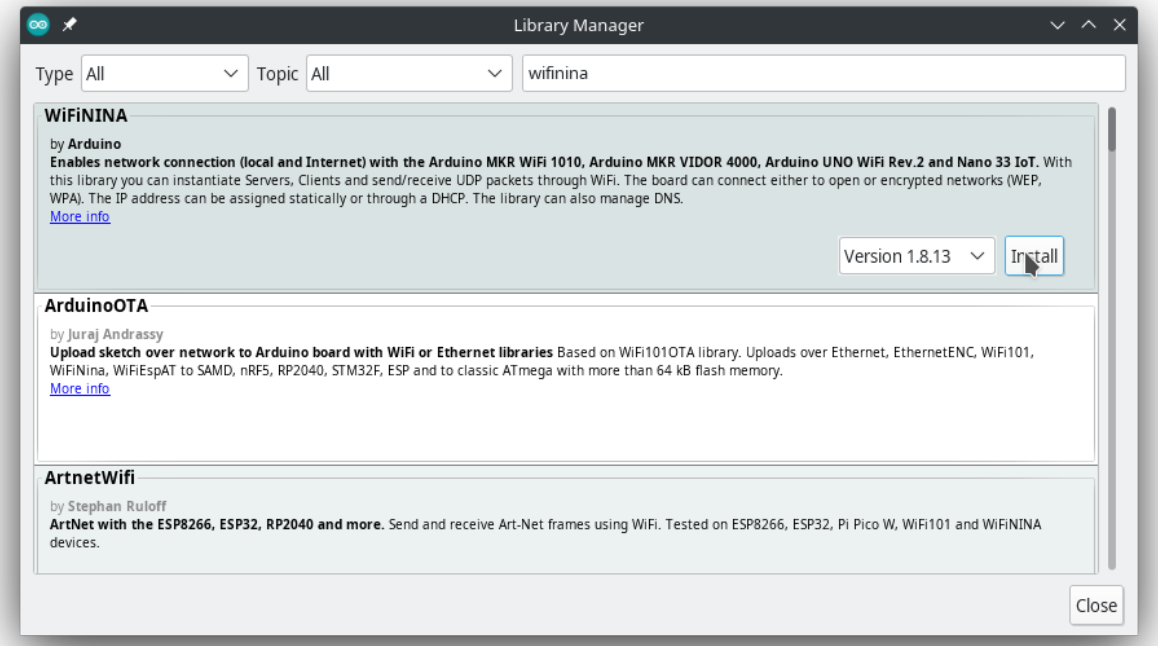

Figure 5 – Installing the WiFiNINA library

Now we need to write some code to loop through a series of colours on the built-in LED. This is shown in code listing 2.

Code listing 2: A simple program to loop through some colours on the built-in RGB LED

```
/*
* 2_rgb_control.ino
*
* this is a simple Arduino program to use pwn to cycle through some
* different colours on the built-in rgb led of the mkr wifi 1010 board
*
* note: this requires the wifinina library to be installed
*
* author: alex shenfield
* date:   30-09-2022
*/

// LIBRARY IMPORTS

// include the wifinina library and the wifi_drv.h file to allow control
// of the built-in led
#include <WiFiNINA.h>
#include <utility/wifi_drv.h>

// INITIAL VARIABLES

// define the leds
const int led_pin_r = 25;
const int led_pin_g = 26;
const int led_pin_b = 27;

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // sets the digital pins as outputs
  WiFiDrv::pinMode(led_pin_r, OUTPUT);
  WiFiDrv::pinMode(led_pin_g, OUTPUT);
  WiFiDrv::pinMode(led_pin_b, OUTPUT);
}
```

```
// this methods loops continuously
void loop()
{
  // full red
  WiFiDrv::analogWrite(led_pin_r, 255);
  WiFiDrv::analogWrite(led_pin_g, 0);
  WiFiDrv::analogWrite(led_pin_b, 0);

  // slight delay
  delay(250);

  // purple
  WiFiDrv::analogWrite(led_pin_r, 128);
  WiFiDrv::analogWrite(led_pin_g, 0);
  WiFiDrv::analogWrite(led_pin_b, 128);

  // slight delay
  delay(250);

  // full blue
  WiFiDrv::analogWrite(led_pin_r, 0);
  WiFiDrv::analogWrite(led_pin_g, 0);
  WiFiDrv::analogWrite(led_pin_b, 255);

  // slight delay
  delay(250);

  // yellow
  WiFiDrv::analogWrite(led_pin_r, 128);
  WiFiDrv::analogWrite(led_pin_g, 128);
  WiFiDrv::analogWrite(led_pin_b, 0);

  // slight delay
  delay(250);

  // full green
  WiFiDrv::analogWrite(led_pin_r, 0);
  WiFiDrv::analogWrite(led_pin_g, 255);
  WiFiDrv::analogWrite(led_pin_b, 0);

  // slight delay
  delay(250);
}
```

Enter this program, upload it to the board, and make sure everything works.

Exercise

Now you should add a potentiometer into the system and use it to fade through the colours using the built-in RGB LED.

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
```

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);
```

Graphically, the output of the LEDs looks something like Figure 6 (below).

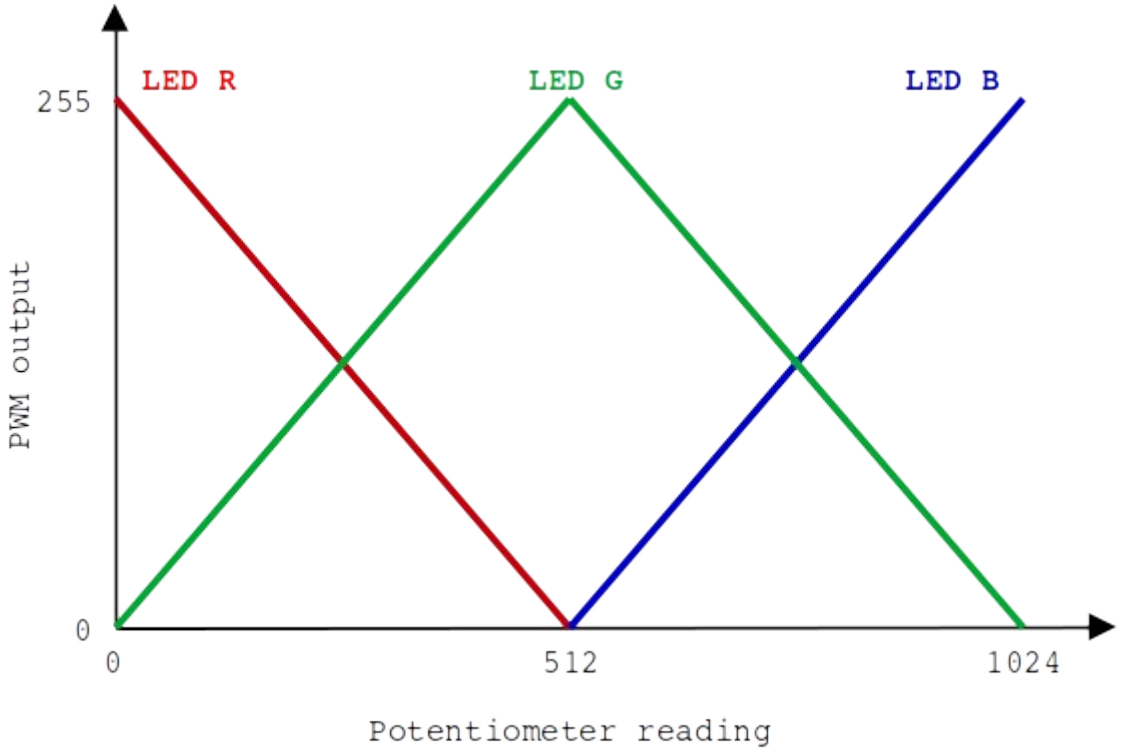

Figure 6 – Graphical representation of LED outputs

Check list

Task	Completed
Task 1 – Program	
Task 2 – Program	
Task 2 – Exercise	

Feedback