Sheffield Hallam University
School of Engineering and Built Environment
BEng (Hons) Electrical and Electronic Engineering
BEng (Hons) Mechanical Engineering

Activity ID		Activity Title			Laboratory Room No.	Level
Lab 102		Basic input and output with Arduino			4302	6
Semester	**Duration [hrs]**	**Group Size**	**Max Total Students**	**Date of approval**	**Lead Academic**	
2	4	1	25	09-24	Alex Shenfield	

Equipment (per student/group)

Number	Item
1	PC
1	Arduino kit

Learning Outcomes

	Learning Outcome
3	Select and apply appropriate embedded networking technologies for a given problem
4	Design, implement and test embedded networked devices

Risk Assessment

General risk assessment for Sheaf 4302 applies (see blackboard for more details).

Basic input and output 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 two extremely important aspects of any kind of embedded system – **input to and output from the system.**

Any kind of embedded microcontroller is only really useful if it has some ability to see the world and react to the environment it is operating in – whether this is 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 and be able to provide both digital and analog signals to control actuators (such as LEDs, motors, speakers, and communication systems).

By the end of this set of laboratory exercises you will understand how to read digital and analog inputs with the with the Arduino MKR WiFi 1010 and process that data to effect the world in some way. You will also understand how to using pulse width modulation to simulate an analog output.

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)!

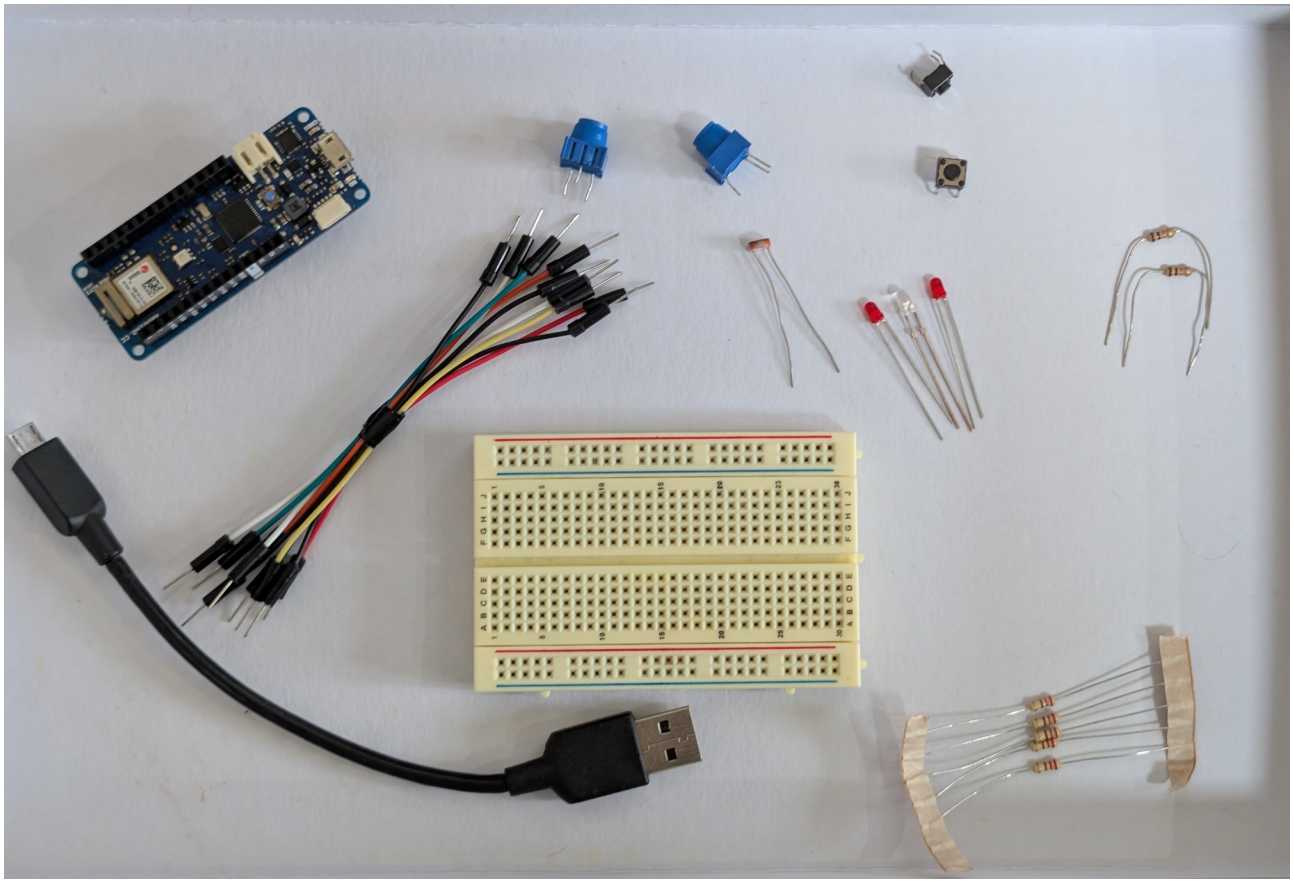


Figure 1 – The necessary equipment for this lab (don't worry if you have LEDs of a different colour or slightly different breadboards)

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 +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. The push-button switch then has one leg connected to +3.3V 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 7 on our Arduino board. We then connect digital pin 6 to the LED.

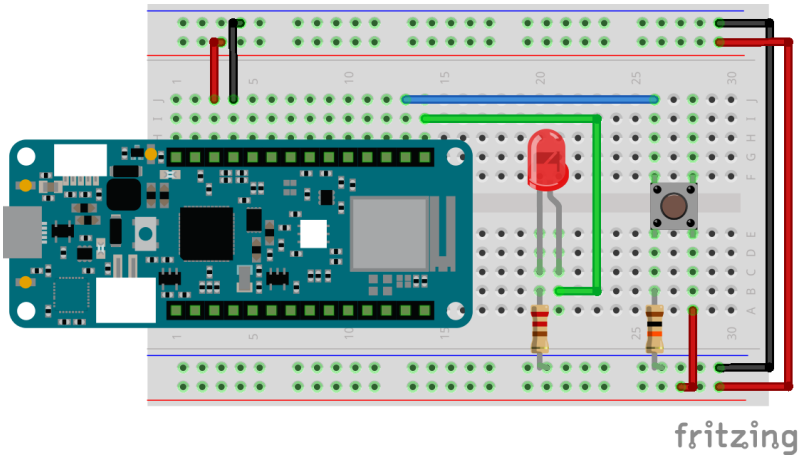


Figure 2 – The switch circuit

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:   2024-11-12
 */

// INITIAL VARIABLES

// 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.

Task 2

We are now going to re-use the circuit we built in task 1, but this time we are going to "latch" the button state (i.e. pressing the button turns the LED on, pressing it again turns the LED off). To do this we 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). This code is shown in code listing 2, below.

Code listing 2: LED latching code

```
/*
 * 2a_led_latch.ino
 *
 * this is a simple Arduino program to use a push button to latch an led (i.e.
 * one press turns it on, the next press turns it off, etc.).
 *
 * 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: prof. alex shenfield
 * date:   2024-11-12
 */

// set pin numbers for button and led
const int button_pin = 7;
const int led_pin    = 6;

// variable for latching the led
int led_state = LOW;

// 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);

  // if the button is pressed toggle the state of the led
  if (button_state == HIGH)
  {
    led_state = !led_state;
    digitalWrite(led_pin, led_state);
  }
}
```

Enter this program and make sure everything works.

You may have noticed that, in the “2a_led_latch” program, the LED sometimes doesn't behave as expected – either not turning on or blinking too quickly. 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.

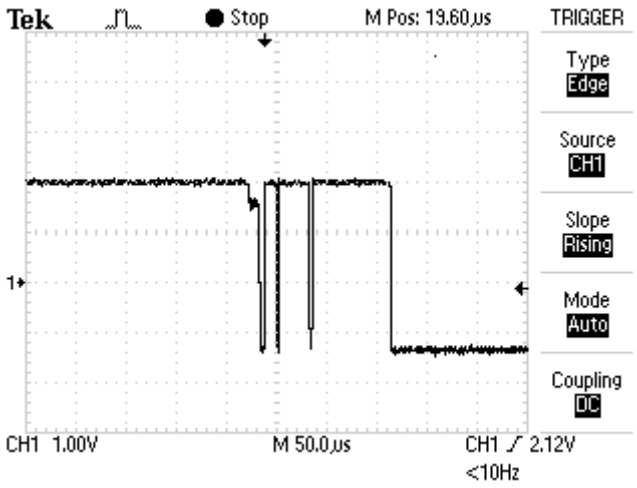


Figure 3 – Oscilloscope trace of a button press

To fix this, we will add “debouncing” to the 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”). A simple debouncing routine is shown in code listing 3, below.

Code listing 3: LED toggle with debouncing code

```
/*
 * 2b_led_latch_with_debouncing.ino
 *
 * this is a simple Arduino program to use a push button to latch an led (i.e.
 * one press turns it on, the next press turns it off, etc.). it now includes
 * added button debouncing.
 *
 * 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: prof. alex shenfield
 * date:   2024-11-12
 */

// set pin numbers for button and led
const int button_pin = 7;
const int led_pin    = 6;

// variable for latching the led
int led_state = 0;

// variables for reading and storing the pushbutton status
int curr_button_state = 0;
int last_button_state = 0;

// set up timing variables for the debounce process
unsigned long debounce_time = 0;
unsigned long debounce_delay = 50;
```

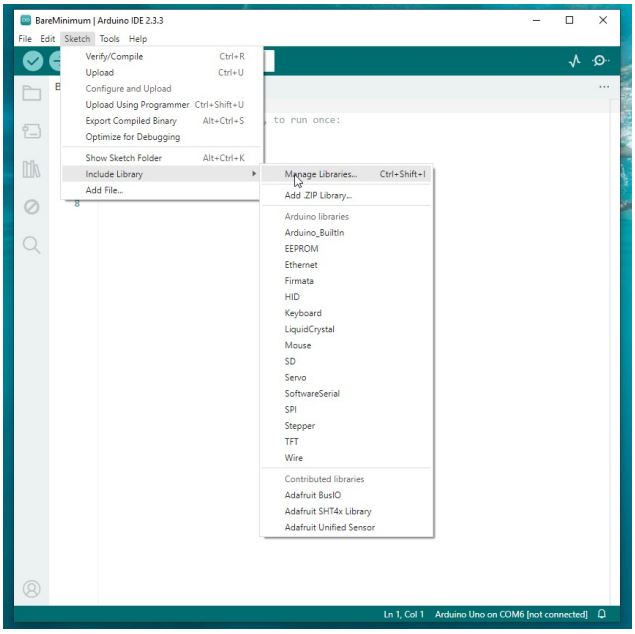
```
// CODE

// 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 to see if the button was pressed - i.e. the input went from LOW to
  // HIGH - and it was long enough after the last transition to not be due to
  // noise

  // if the button state was changed, we reset the timer
  if (button_state != last_button_state)
  {
    debounce_time = millis();
  }

  // if this was a genuine button push and not noise (measured by the button
  // state having changed for longer than the debounce delay)
  if ((millis() - debounce_time) > debounce_delay)
  {
    // if the button state has changed, update the button state. if it has
    // gone from LOW to HIGH then also toggle the state of the led
    if (button_state != curr_button_state)
    {
      curr_button_state = button_state;
      if (curr_button_state == HIGH)
      {
        led_state = !led_state;
        digitalWrite(led_pin, led_state);

        // this is where we will do whatever else we want to happen after
        // pressing the button!
      }
    }
  }

  // update the button state (right at the end of the loop!)
  last_button_state = button_state;
}
```

Enter this program and make sure everything works.

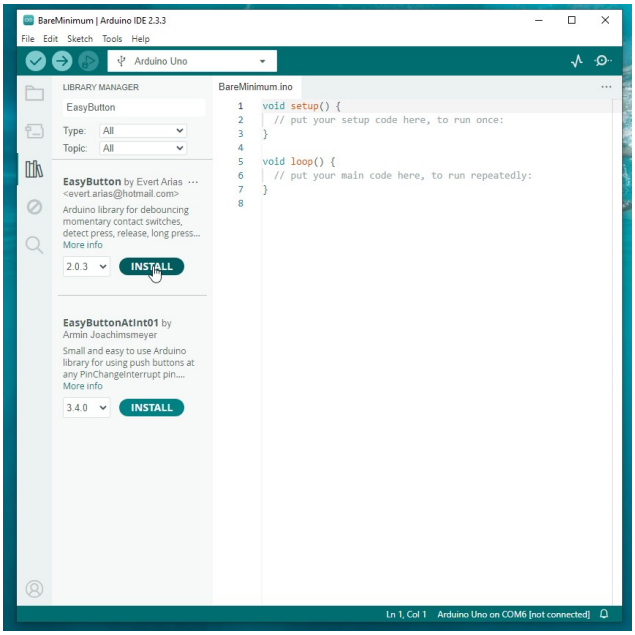
Task 3

In the previous task we have seen how button contacts can "bounce" leading to false indications of button presses. We also developed our own debouncing code based on timings – however, this code is difficult to extend to multiple buttons. In this task we will use the EasyButton library by Evert Arias[1].

We can install this using the "Manage Libraries ..." option in the Sketch menu (see Figure 4 below). Once the "manage libraries" tab opens on the left hand side, you can search for "EasyButton".



a) Selecting "Manage Libraries ..."



b) Searching for the "EasyButton" library and installing it

Figure 4 – Installing a library using the Library Manager

The EasyButton library allows us to not only reduce the amount of extra code we need to write to support button debouncing, but also provides a whole host of extra features such as:

1. Simplified support for multiple buttons
2. Support for interrupts
3. Support for double click and long click events

1 https://github.com/evert-arias/EasyButton

A simple debouncing example using this library and the circuit from Figure 5 is shown in code listing 4, below.

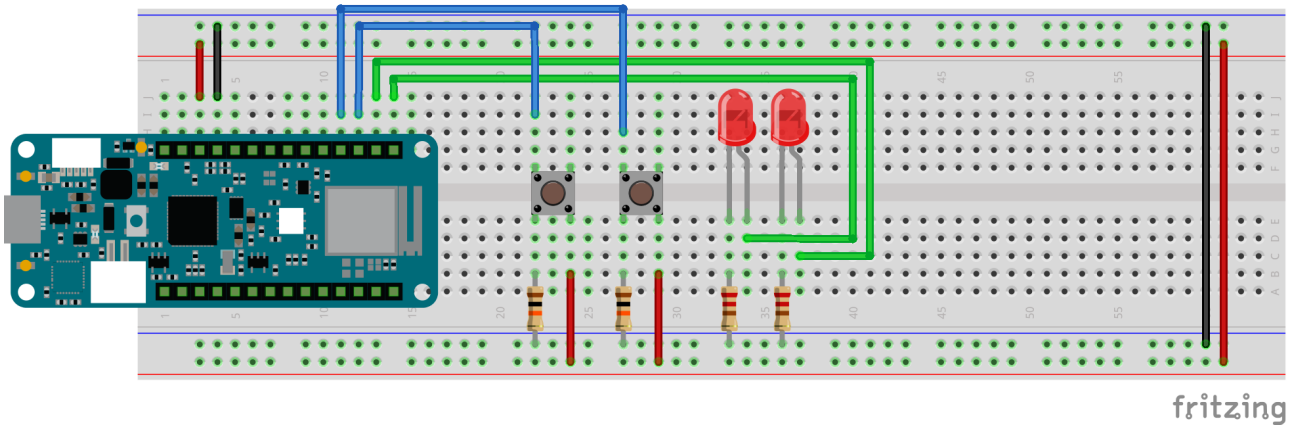


Figure 5 – The two switch circuit

Code listing 4:

```
/*
 * 3_lazy_button_handling.ino
 *
 * a simple example of using the EasyButton library to handle button presses
 * without a loads of extra code. this provides all the functionality in the
 * previous debouncing and latching examples, just without the need to manage
 * timings ourselves.
 *
 * this program uses pins 8 and 9 for the push buttons and pins 6 and 7 for the
 * leds (pin 6 is the built in led on the MKR series of boards).
 *
 * author: prof. alex shenfield
 * date:   2024-11-12
 */

// LIBRARY IMPORTS

// include our lazy button library
#include <EasyButton.h>

// BUTTON AND LED DECLARATIONS

// light bulb settings
const int light_bulb_1 = 6;
const int light_bulb_2 = 7;
int light_state_1 = LOW;
int light_state_2 = LOW;

// light switch 1 (on pin 8) and 2 (on pin 9)
EasyButton light_switch_1(8);
EasyButton light_switch_2(9);
```

```
// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // set up serial comms for debugging
  Serial.begin(9600);
  while (!Serial);
  Serial.println("starting lazy button handler ...");

  // set up the first light bulb and light switch (including the button
  // pressed callback function)
  pinMode(light_bulb_1, OUTPUT);
  light_switch_1.begin();
  light_switch_1.onPressed(light_toggle_1);

  // set up the second light bulb and light switch (including the button
  // pressed callback function)
  pinMode(light_bulb_2, OUTPUT);
  light_switch_2.begin();
  light_switch_2.onPressed(light_toggle_2);
}

// this methods loops continuously
void loop()
{
  // read the buttons statuses
  light_switch_1.read();
  light_switch_2.read();
}

// APPLICATION FUNCTIONS

// callback function attached to the first light switch button
void light_toggle_1()
{
  // print a status message
  Serial.println("button 1 pressed!");

  // toggle the state of the light
  light_state_1 = !light_state_1;
  digitalWrite(light_bulb_1, light_state_1);
}

// callback function attached to the second light switch button
void light_toggle_2()
{
  // print a status message
  Serial.println("button 2 pressed!");

  // toggle the state of the light
  light_state_2 = !light_state_2;
  digitalWrite(light_bulb_2, light_state_2);
}
```

Enter the code above, and test that everything works!

Task 4

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 6.

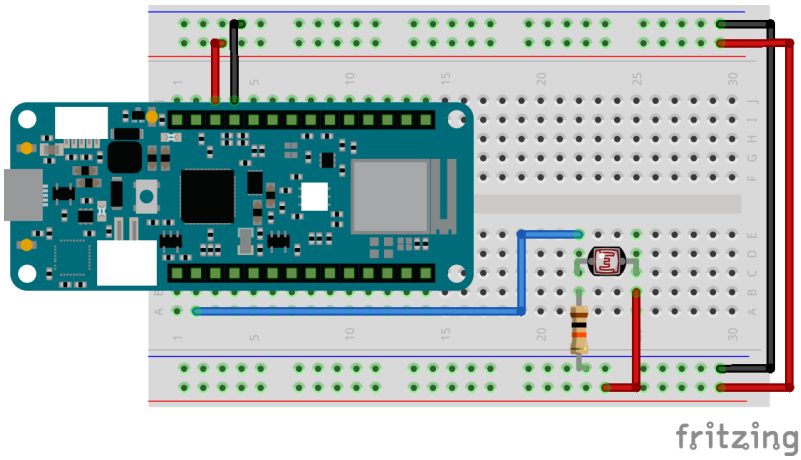


Figure 6 – The LDR circuit

Code listing 5 then shows a simple program to read this voltage value and display it on the Arduino IDE Serial Monitor.

Code listing 5: Code to read the LDR value and display it on the Serial Monitor

```
/*
 * 4_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: prof. alex shenfield
 * date:   2024-11-12
*/

// INITIAL VARIABLES

// 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 7.

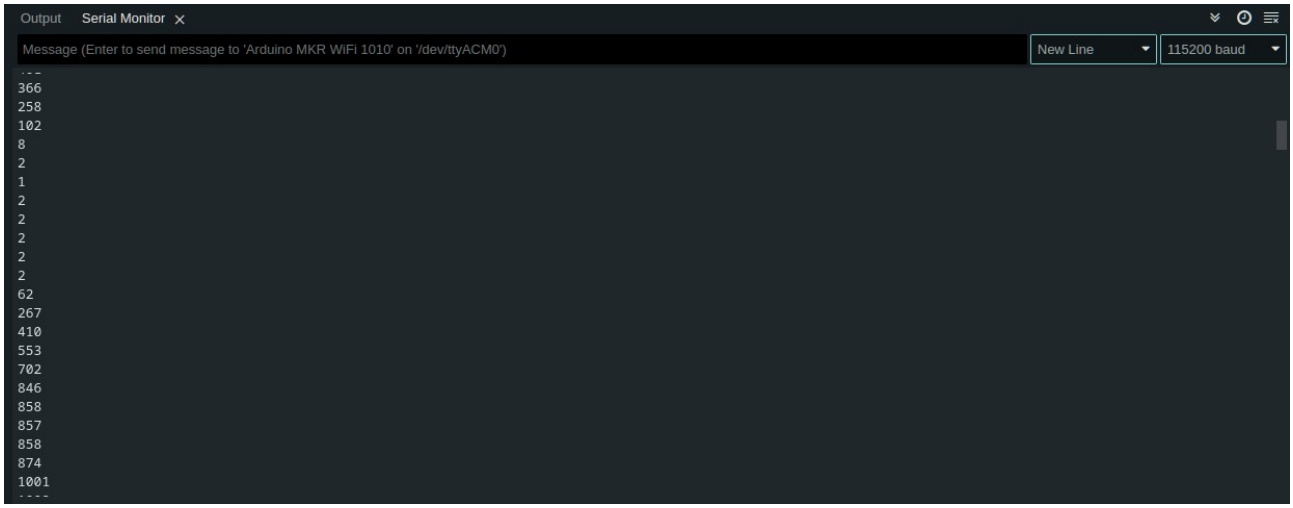


Figure 7 – Light data displayed on the Arduino Serial Monitor

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[2] to graph these light values (see Figure 8)!

Figure 8 – Plotting light data via the Arduino Serial Plotter

2 This can be found under "Tools > Serial Plotter".

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 9 for an example).

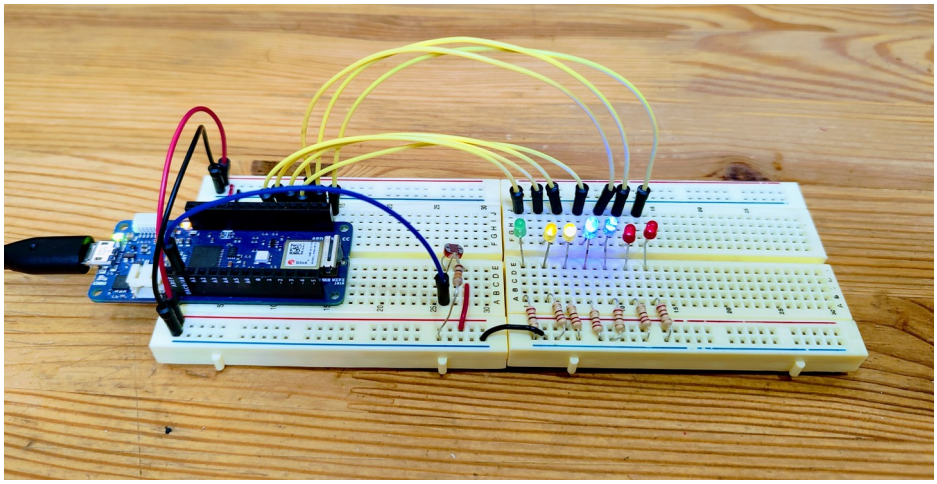


Figure 9 – An example light sensitive LED bar graph

Task 5

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 10 (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).

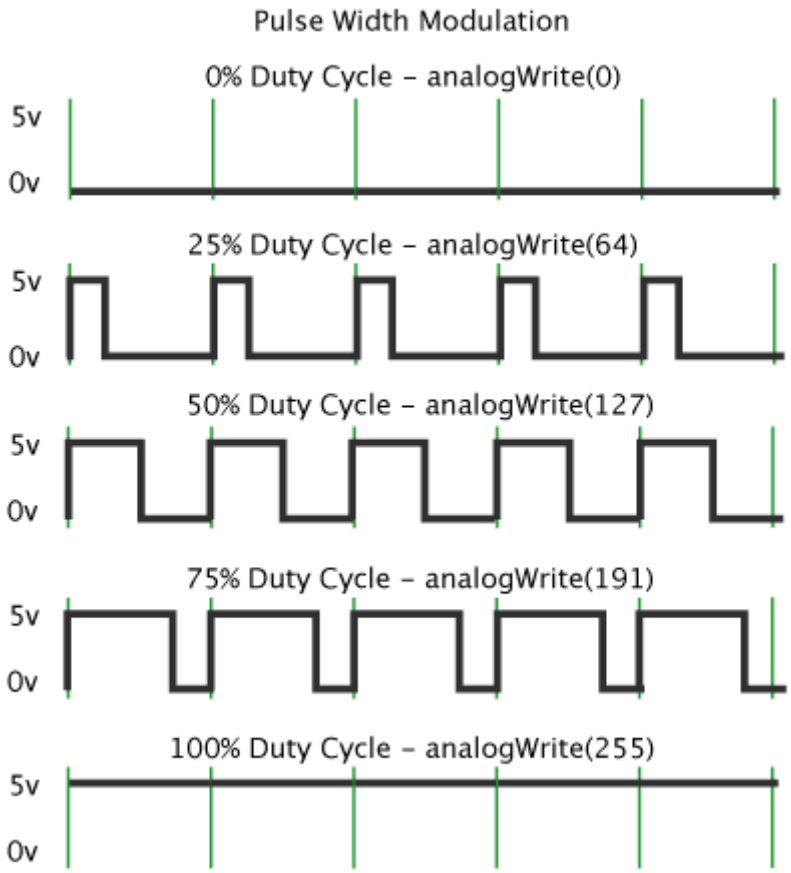


Figure 10 – Illustration of pulse width modulation[3]

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.

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

In this task we will use the analogRead function (as we did in task 4) to read the value of a potentiometer and then use pulse width modulation to dim an LED.

Firstly create the circuit shown in Figure 11, 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).

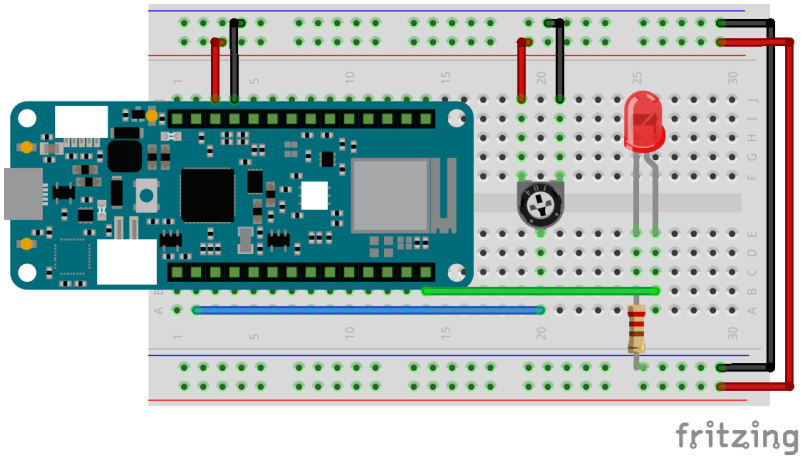


Figure 11 – LED dimmer circuit

See code listing 1 for the complete program.

Code listing 6: Code to dim an LED based on the value of a potentiometer

```
/*
 * 5_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: prof. alex shenfield
 * date:   2024-11-12
 */

// 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 6

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

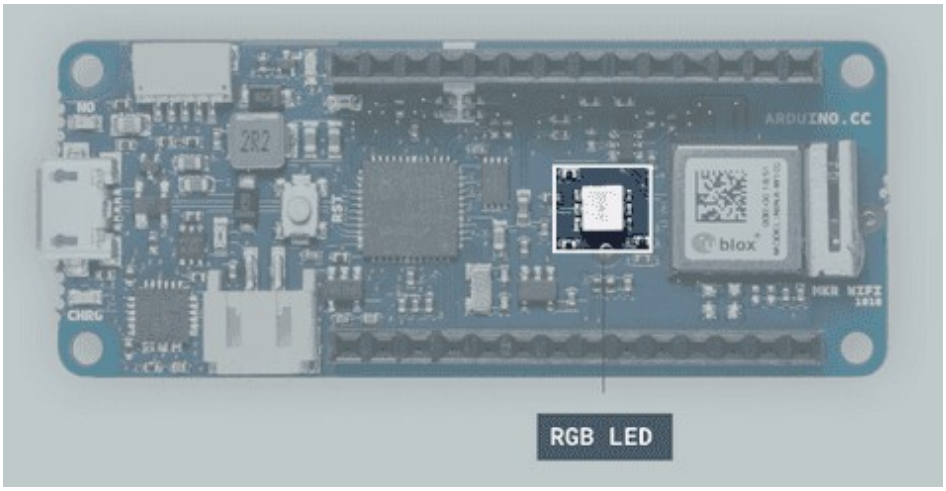


Figure 12 – The built-in RGB LED on the Arduino MKR WiFi 1010

To control this built-in LED, we'll actually have to use the WiFiNINA library[4] (as well as using PWM to change the colours). Figure 13 shows the installation of this library via the Arduino library manager.

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

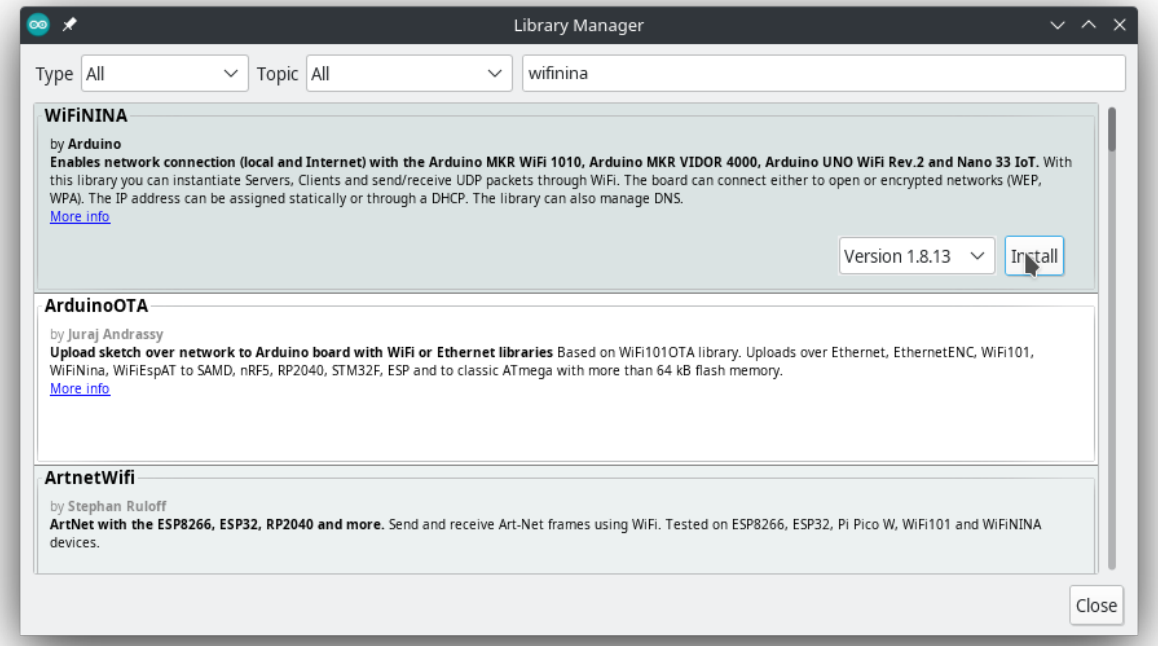


Figure 13 – Installing the WiFiNINA library

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 7.

Code listing 7: A simple program to loop through some colours on the built-in RGB LED

```
/*
 * 6_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: prof. alex shenfield
 * date:   2024-11-12
*/

// 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.

Check list

Task	Completed
Task 1	
Task 2	
Task 3Task 4	
Task 4	
Exercise 4a	
Exercise 4b	
Task 5	
Task 6	

Feedback