Sheffield Hallam University
School of Engineering and Built Environment

Activity ID	Activity Title	Laboratory Room No.	Level
Lab 2	Analog and Digital Input 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 and Digital Input 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 – **input to the system.**

Any kind of embedded microcontroller is only really useful if it has some ability to see the world – whether 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.

By the end of this set of laboratory exercises you will understand how to read both analog and digital data with the Arduino microcontrollers and process that data to effect the world in some way.

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)!

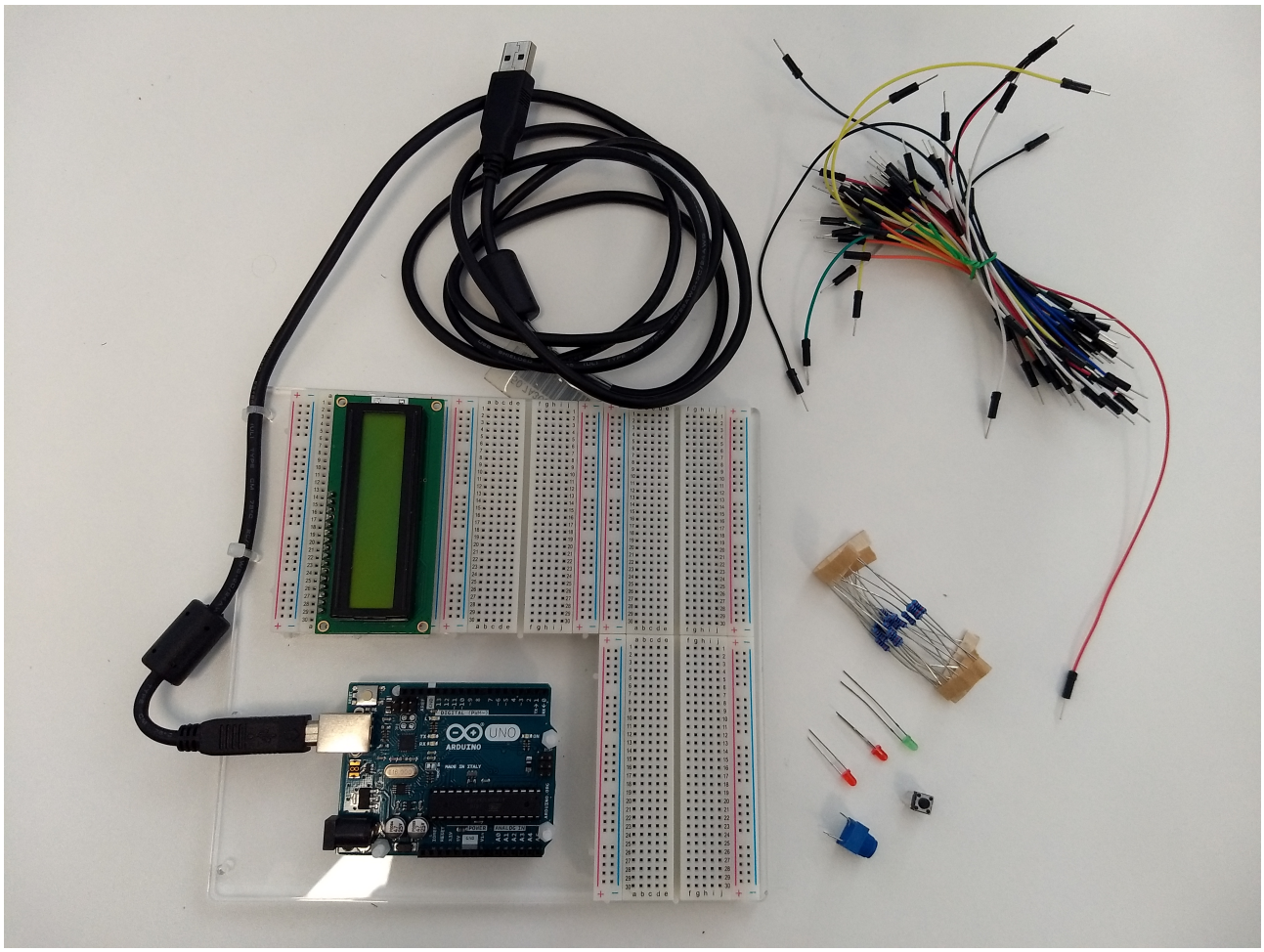


Figure 1 – The necessary equipment for this lab (don't worry if you have LEDs of a different colour or slightly different breadboards)

Task 1

In this task we are going to use a simple pushbutton 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 +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.

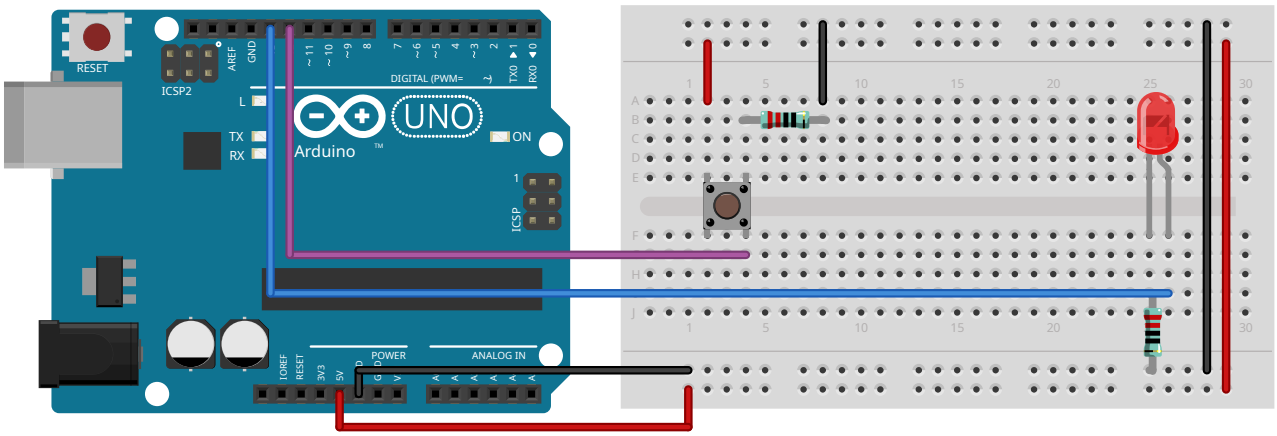


Figure 2 – The switch circuit

Q1 What is a “pull-down” resistor?

Q2 We could also have a “pull-up” resistor. What is the difference?

Q3 Why do we need “pull-up” or “pull-down” resistors?

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 1 for a simple program to accomplish this.

Code listing 1: LED on/off code

```
/*
 * 1_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
 *
 * author: prof. alex shenfield
 * date:   2024/11/12
 */

// 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.

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
 *
 * latch an led based on the press of a button
 *
 * author: prof. alex shenfield
 * date:   2024/11/12
 */

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

// 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 “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.

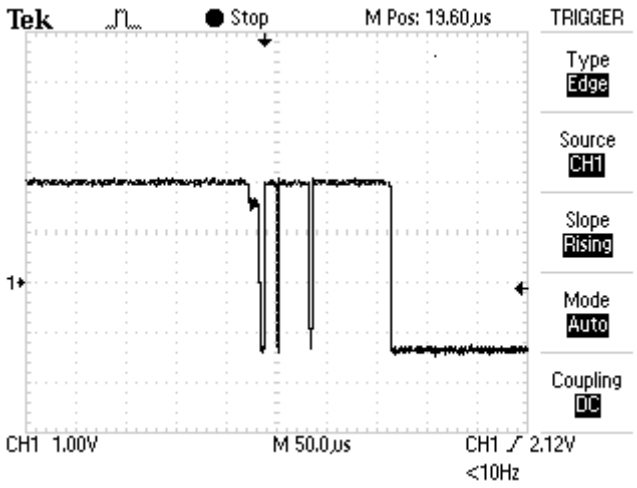


Figure 3 – Oscilloscope trace of a button press

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_on_off_with_debouncing.ino
 *
 * latch an led based on the press of a button (now with added button
 * debouncing)
 *
 * author: prof. alex shenfield
 * date:   2024/11/12
 */

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

// 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;
```

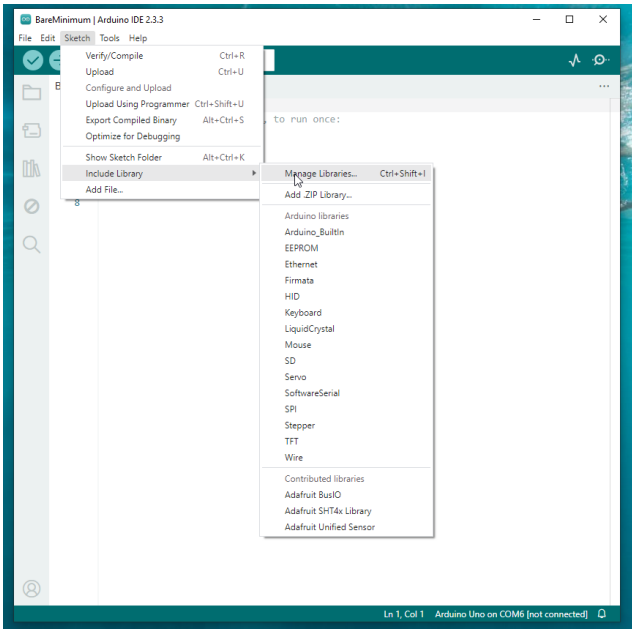
```
// CODE

// 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.

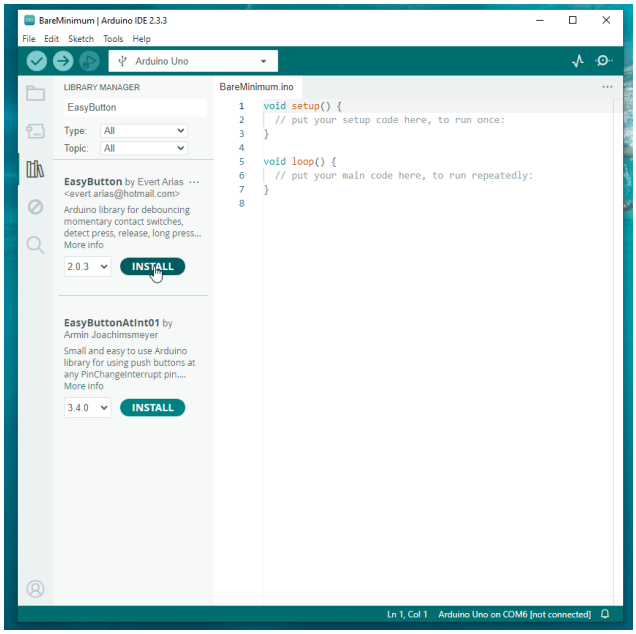
Task 3

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

Now build the two button and two LED circuit shown in Figure 5, below.

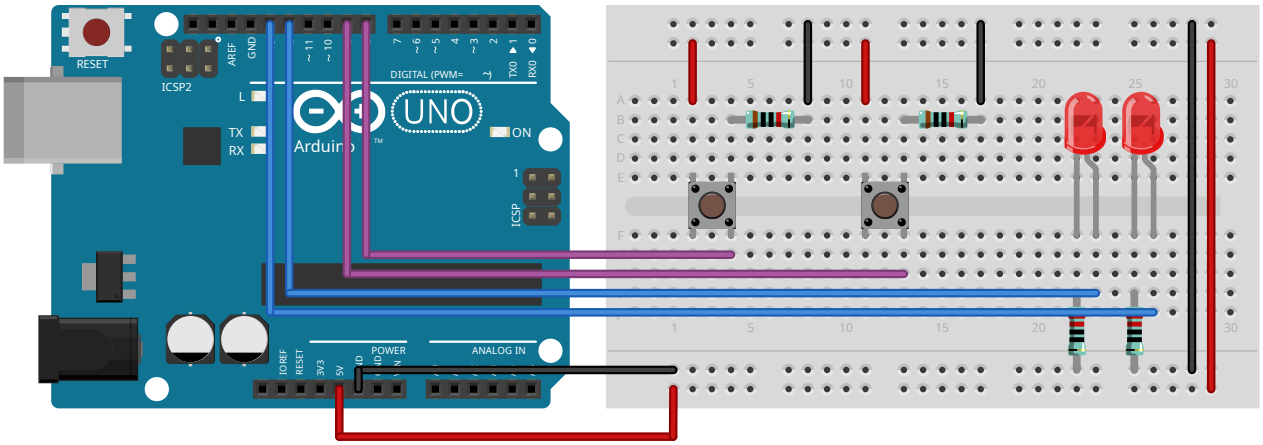


Figure 5 – The two button switch circuit

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

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

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

Code listing 4:

```
/*
 * 3_lazy_button_handling.ino
 *
 * a simple example of using the EasyButton library to handle button presses
 * without a tonne of extra code
 *
 * 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 = 13;
const int light_bulb_2 = 12;
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 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.

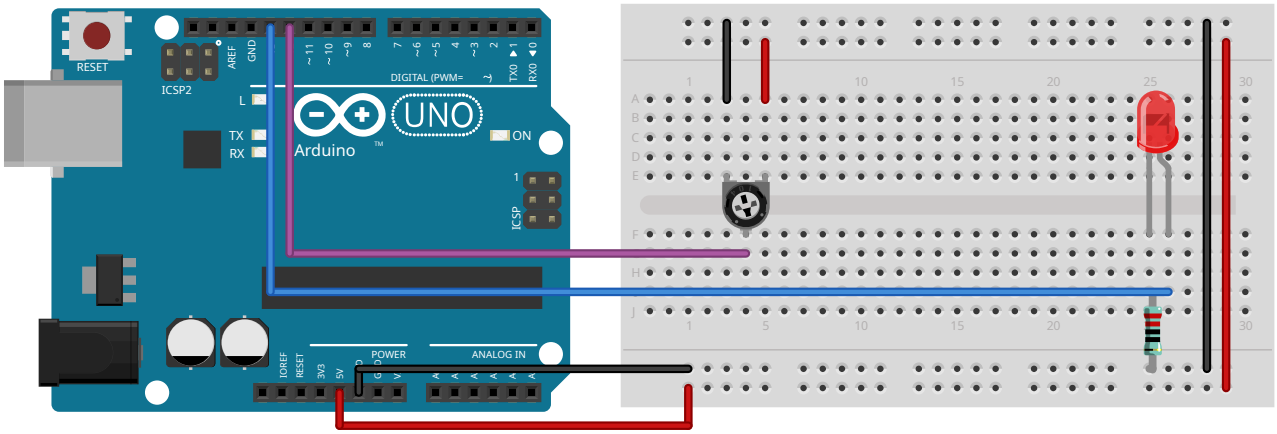


Figure 6 – The dimmer switch circuit

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 13.

We then need to write our program to control the LED (see code listing 5).

Code listing 5: Controlling the blink rate of an LED

```
/*
 * 4_analog_blink_rate.ino
 *
 * simple program to adjust the blink rate of an led based upon the value of a
 * potentiometer
 *
 * author: prof. alex shenfield
 * date:   2024/11/12
 */

// set pin numbers
const int led_pin = 13;
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.

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?

Check list

Task 1 – Q1	
Task 1 – Q2	
Task 1 – Q3	
Task 1 – Program	
Task 2 – Latch program	
Task 2 – Debouncing program	
Task 3 – Program	
Task 4 – Program	
Task 4 – Q1	
Task 4 – Q2	

Feedback/ reflections