

Arduino Introduction Activity

Authors: Punit Shah, Karnan Tamilchelvan, Benjamin Zaionz, Avi Bhadore, Michelle Chen, Silas Ifeanyi, Michael McPhee, Nancy Nelson, Chris Rennick

Table of Contents...

Contents

Microcontrollers	4
Required Materials	4
1 Activity – Digital Inputs and Outputs	4
1.1 Background.....	4
1.1.1 Breadboard	4
1.1.2 Wiring Conventions	5
1.1.3 Digital Signals.....	5
1.1.4 Arduino Uno Board	6
1.1.5 Arduino Pinout Map	6
1.1.5 Arduino Integrated Development Environment (IDE)	7
1.1.6 LED Polarity.....	8
1.1.7 Pull-up and Pull-down Resistors	8
1.2 Task.....	9
1.2.1 Build your Circuit.....	9
1.2.2 Create Your Project (Sketch)	11
1.2.3 Add your Code.....	11
1.2.5 Test your Circuit	12
1.2.6 Aside – Multimeters	12
1.2.6a - Measuring Resistance.....	12
1.2.6b - Measuring Voltage	13
1.2.6c - Measuring Current	14
1.3 Challenge 1	14
2 Activity – Analog Inputs and UART	15
2.1 Background.....	15
2.1.1 Analog Signals	15
2.1.2 Measuring Analog Signals using Microcontrollers – ADCs	15
2.1.3 Potentiometers.....	15
2.1.4 UART and Serial Communication	16

2.2 Task.....	16
2.2.1 Wire your Circuit.....	16
2.2.2. Add your Code	17
2.3 Challenge 2	17
3. PWM and Oscilloscopes	17
3.1. Background.....	18
3.1.1. PWM.....	18
3.1.2. Oscilloscopes.....	19
3.2. Task	19
3.3.1. Wire your Circuit	19
3.3.2. Add your Code	19
3.3.3. View the PWM signal.....	20
3.3. Challenge 3.....	21
4. Related Resources	21

Microcontrollers

Microcontrollers are small one-chip computers that are at the core of much of today's technology to automate and control tasks. Generally, they have a Central Processing Unit (CPU), some memory, and programmable input and output ports. This workshop introduces you to the essentials of using an Arduino Uno, a popular microcontroller. You'll learn how to build and control simple digital and analog circuits, send and receive data signals between devices, and generate Pulse Width Modulation (PWM) signals to control external devices. In the process you'll use breadboards, meters and scopes.

Required Materials

- 2 Switches (Push Button)
- 2 LEDs
- 1 Breadboard
- 2 Resistors (220 Ω)
- 1 Resistor (10 k Ω)
- 1 Potentiometer
- 1 Oscilloscope
- 1 Multimeter
- 1 Arduino Uno
- Assorted Wires

1 Activity – Digital Inputs and Outputs

The ability to read and send digital signals is fundamental in nearly all electronic devices. Controlling an LED with a pushbutton might seem simple, but this activity helps you grasp the principles of digital communication between a microcontroller and external components. In real-world applications, digital inputs and outputs are used in everyday systems, from simple appliances to complex control systems in cars or industrial machines.

1.1 Background

1.1.1 Breadboard

A breadboard (see Figure 1) is a prototyping tool used for building and testing circuits without the need for soldering. It consists of rows and columns of conductive strips underneath a plastic surface with holes, allowing you to easily insert components and wires to form electrical connections. A typical breadboard is divided into two main areas: the power rails and the terminal strips.

The power rails, running along the sides, are used for connecting power (e.g., 3.3V or GND) throughout your circuit. The terminal strips in the center are arranged in rows and connect vertically underneath the board, making it possible to link components without direct wiring. Figure 1 shows how the connections are made internally on a typical breadboard.

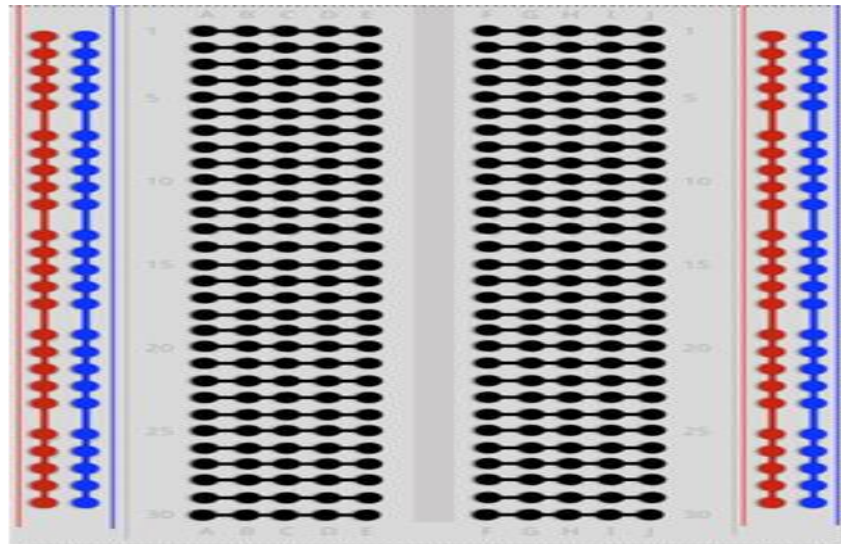


Figure 1 : Breadboard Connections

1.1.2 Wiring Conventions

When wiring circuits or creating schematics, it's helpful to follow color conventions to make your work easier to read, debug, and share. A common standard is to use red wires for power (Vcc) and black wires for ground (GND).

Beyond that, it's up to you to define your own consistent system. A good practice is to use one wire color per signal or per section of a circuit, which keeps things organized — especially as circuits grow more complex.

When you're just starting out, it's okay to use a variety of colors to help visualize how different parts of the circuit connect and how signals flow. Over time, developing your own consistent wiring style will make building and troubleshooting much easier.

1.1.3 Digital Signals

Electrical signals fall into two main categories: digital and analog. A digital signal has only two possible states — known as “High” and “Low”. On the Arduino Uno, a High signal is 5 volts, and a Low signal is 0 volts, also called ground (GND).

Digital signals are used throughout electronics for on/off logic, and they form the foundation for digital communication and control. We'll explore analog signals, which can vary smoothly between voltages, in the next section.

1.1.4 Arduino Uno Board

The Arduino UNO (see Figure 2) is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins and 6 analog inputs. It's an easy-to-use system designed for hobbyists.



Figure 2: Arduino Uno

1.1.5 Arduino Pinout Map

A pinout map shows the available pins on your Arduino Uno microcontroller and what functions those pins can be used for. Each pin can have multiple uses:

- Digital I/O (Input/Output) allows the Arduino Uno to read and write to components (LEDs, buttons, etc.) via digital signals. Digital output pins can send signals, either HIGH (5V) or LOW (0V), to other components. Digital input pins can read the state of a digital signal, such as whether a button is pressed or not. In this activity you'll use a digital output to turn an LED on and off, and a digital input to detect whether a button is pressed.
- Analog inputs read continuous voltage from sensors or potentiometers.
- Communication protocols allow the Arduino Uno to communicate with peripherals via I2C, SPI and UART. Additional info on these protocols is available in section 2.

Figure 3 shows the pinout map for the Arduino Uno board you're using in this activity. You will likely need to refer to this figure throughout the activity. The digital I/O pins are labeled D0 – D15, and the analog inputs are labeled A0 – A5. There are also pins that are dedicated to the communication protocols labeled I2C, SPI and UART.

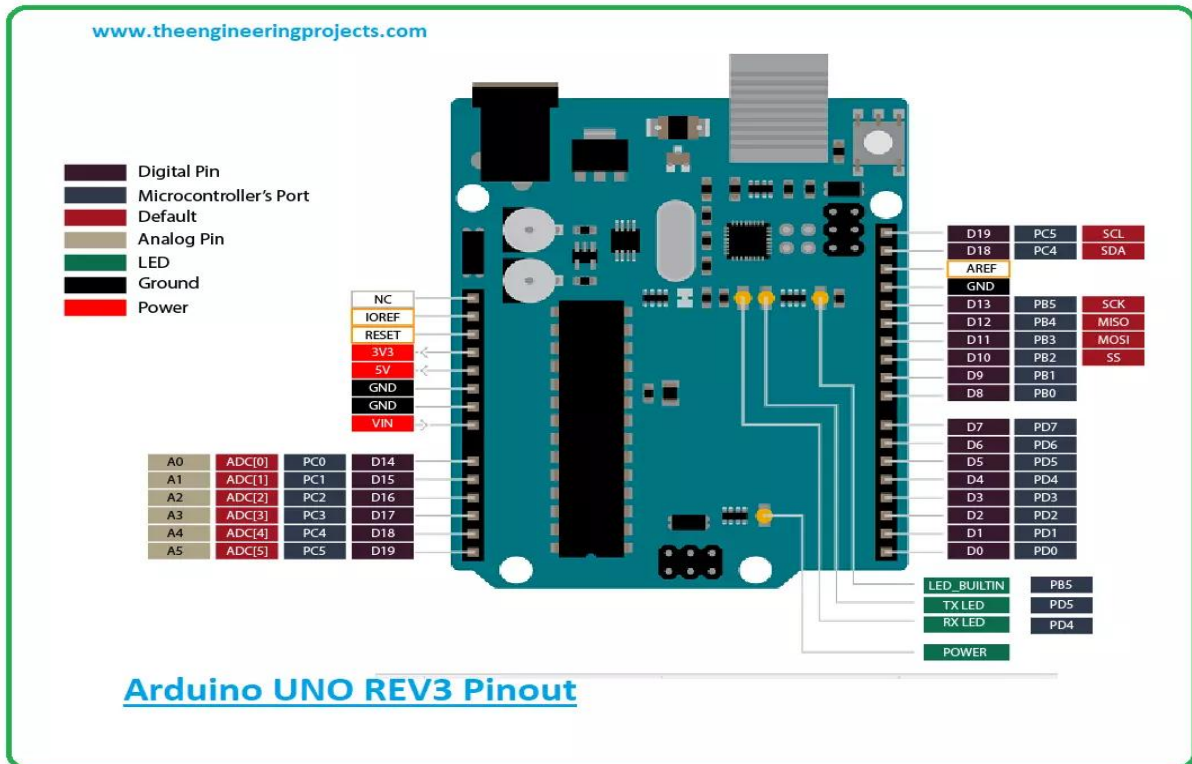


Figure 3- Arduino Uno Pinout Map

1.1.5 Arduino Integrated Development Environment (IDE)

If you haven't installed the Arduino's IDE yet, download it from the [official Arduino website](https://www.arduino.cc/en/software) (see Figure 4). Follow the setup instructions that pop up when you open the file.



Figure 4 – Downloading Arduino's IDE

Start the application and do the following:

1. Select **Tools > Boards > Boards Manager**
2. Search for "**arduino uno r4**"
3. Select **Install** and accept all prompts for installation
4. Once the installation is complete, please confirm you see the following "**Tools > Boards > Arduino UNO R4 Boards > Arduino UNO R4 Minima**"

1.1.6 LED Polarity

Many electronic components, such as resistors, are non-polar. This means that the orientation of the component relative to the flow of electricity does not matter. Other components are polar, where the orientation of the component relative to the flow of electricity does matter. One such component is a diode. Diodes are designed specifically to allow electricity to flow in a certain direction and not in the other. LEDs (light emitting diodes) are diodes that also emit light. Figure 4 shows an LED.

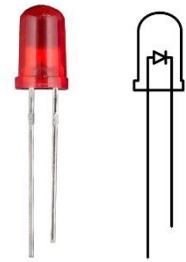


Figure 5- LED

Notice that one leg is longer than the other. The longer leg is the anode and is connected to a positive voltage. The shorter leg is the cathode and is connected to a negative voltage (GND). If the LED is connected in the wrong orientation, then no electricity will flow, and the LED will not light up.

1.1.7 Pull-up and Pull-down Resistors

If a line in a circuit isn't explicitly set to a voltage it is considered "floating". In electronics, it is undesirable for any lines to be floating, so we use pull-up and pull-down resistors to passively set the voltage of lines that are not being set otherwise. Figure 6 shows the wiring diagram for pull-up and pull-down resistors. Please note that MCU stands for microcontroller unit, which in our case is the Arduino Uno.

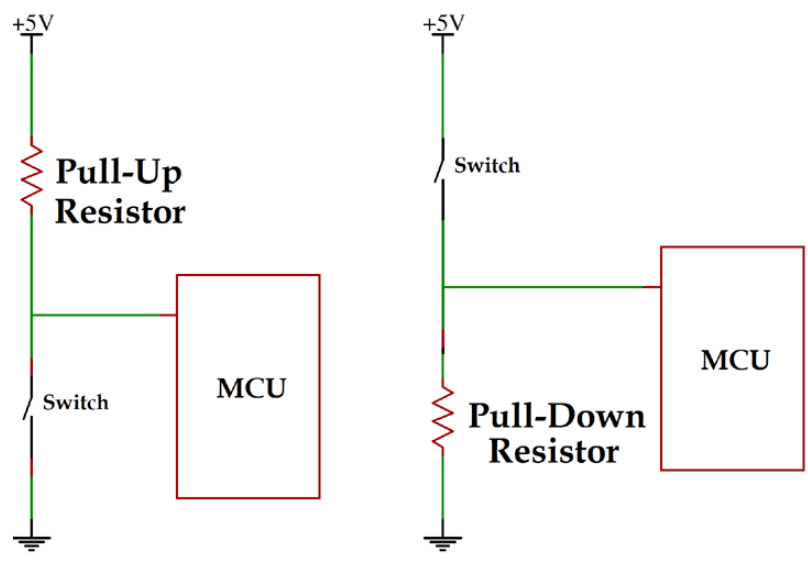


Figure 6- Pull-up and Pull-down Resistors

Consider the pull-down resistor diagram. When the switch is open, the voltage on the MCU pin is very close to ground (if the pull-down resistor has a low resistance, which they typically do). When the switch is closed, the voltage on the MCU pin is 5V, and the resistor measures 5V across. This behaviour is mirrored for the pull-up resistor.

1.2 Task

In this task you will control an LED with a button. When the button is pressed, the LED will turn on, and when the button is released, the LED will turn off. Figure 7 shows the circuit you're going to build:

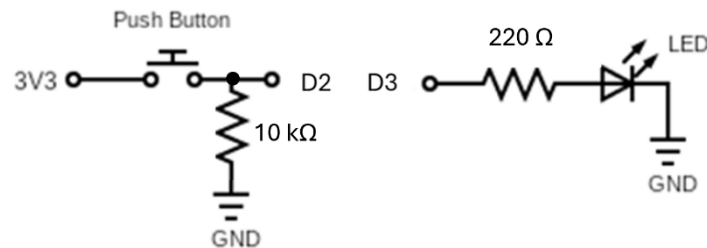


Figure 7 – LED – pushbutton circuit

Here's how the circuit works:

1. The button is connected between the 5V power rail and a digital input pin (e.g., pin 2). When the button is pressed, the pin is connected directly to 5V, and the Arduino will read the pin as “High”.
2. When the button is not pressed, the pin is tied to ground through the 10kΩ pull-down resistor between the input pin and GND. This ensures the input pin not floating when the button is not pressed.
3. The LED is connected in series with a current-limiting resistor to prevent it from burning out. The resistor connects to a digital output pin, which can be toggled HIGH or LOW to turn the LED on or off. Note that the LED must be oriented correctly, with the longer leg (anode) connected to the resistor (and then to the digital output), and the shorter leg (cathode) connected to GND. If you flip the LED, it will not light up.

1.2.1 Build your Circuit

If you've never used a breadboard before, please refer to Figure 8 and follow these instructions:

1. Place the push button on the breadboard straddling the centre divider
 - a. Connect one of its legs to the breadboard's power rail with a red wire
 - b. Connect its other leg to one leg of the 10 kΩ resistor (brown, black, orange). Put the resistor's other leg in a different row
 - c. Connect the resistor's other leg to the breadboard's ground rail with a black wire
2. Place the LED on the breadboard
 - a. Connect the LED's cathode (shorter leg) to the breadboard's ground rail (black wire)

- b. Connect the LED's anode (longer leg) to one end of the 220 Ω resistor (red, red, brown). Put the resistor's other leg in a different row
 3. Connect the circuit to the Arduino Uno
 - a. connect the other pushbutton lead to digital pin 2
 - b. connect the other leg of the 220 Ω resistor to digital pin 3
 - c. connect the breadboard's power rail to the 5V pin on the Arduino with a red wire
 - d. connect the ground rail to the Arduino's GND pin with a black wire
 4. Double check your connections
 - a. Arduino's 5V to the power rail
 - b. Breadboard's power rail to the pushbutton
 - c. Pushbutton to 10 k Ω resistor
 - d. 10 k Ω resistor to digital pin 2
 - e. Digital pin 3 to the 220 Ω resistor
 - f. 220 Ω resistor to the LED's anode (longer leg)
 - g. LED's cathode (shorter leg) to the breadboard's ground rail
 - h. Ground rail to Arduino's GND

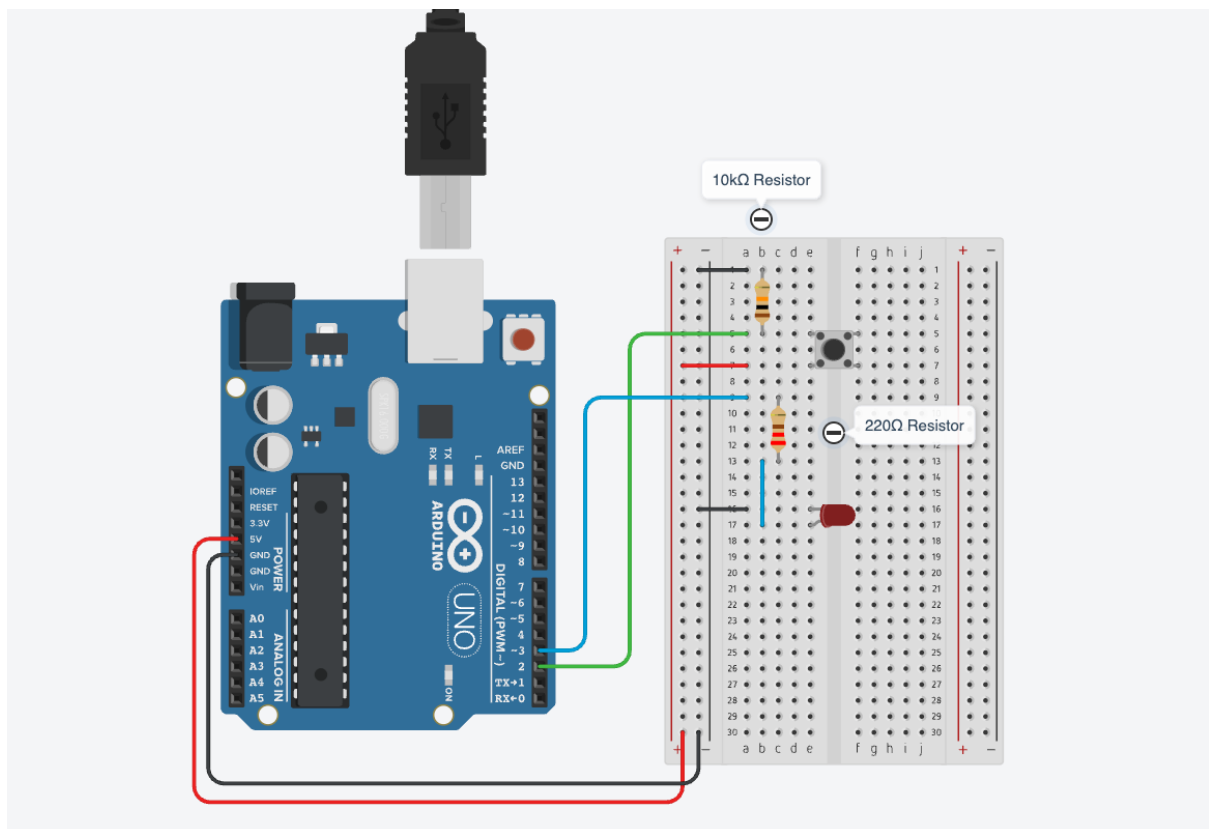


Figure 8 – Sample Layout for Pushbutton and LED

1.2.2 Create Your Project (Sketch)

Now you're ready to create the project (aka sketch) and write the code to control your system. Follow these steps:

1. Start the Arduino Integrated Development Environment (IDE)
2. Go to **File > New** to create a new sketch (project). This will open a blank window where you can begin coding for the Arduino Uno (see **Error! Reference source not found.**).
3. Click **File > Save As** and name your project (i.e. *learningArduino.ino*). It's a good habit to avoid spaces or special characters in the file name.

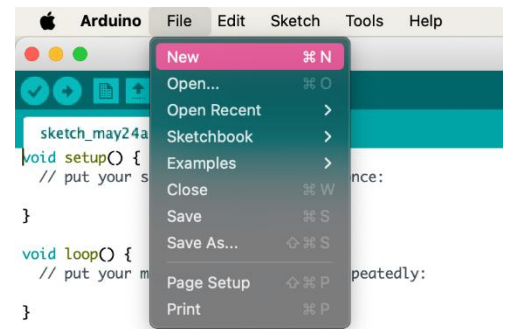


Figure 9– Creating a New Project

1.2.3 Add your Code

Now it's time to write your code. We've written it for you this time, so please copy and paste this code into the Arduino IDE:

```
-----  
  
const int buttonPin = 2;  
  
const int ledPin = 3;  
  
void setup () {  
    pinMode(buttonPin, INPUT);  
    pinMode(ledPin, OUTPUT);  
}  
  
void loop() {  
    int state = digitalRead(buttonPin);  
    if (state == HIGH) {  
        digitalWrite(ledPin, HIGH);  
    } else {  
        digitalWrite(ledPin, LOW);  
    }  
}  
  
-----
```

Here's an explanation of the code:

4. `const int buttonPin = 2;`
 - a. Declares a constant variable for the button input pin. This means digital pin 2 will be used to detect the state of the pushbutton.

5. `const int ledPin = 3;`
 - a. Declares a constant variable for the LED output pin. This sets digital pin 3 as the one that controls the LED.
6. `pinMode(buttonPin, INPUT);`
 - a. Configures pin 2 as a digital input, allowing it to detect if the button is pressed or not.
7. `pinMode(ledPin, OUTPUT);`
 - a. Configures pin 3 as a digital output, which can be set HIGH (5V) or LOW (0V) to control the LED.
8. `digitalRead(buttonPin);`
 - a. Checks the voltage level of pin 2. If the button is pressed, it reads HIGH; if not pressed, it reads LOW.
 - b. If the button is pressed, `digitalWrite(ledPin, HIGH);` turns the LED on.
 - c. If the button is not pressed, `digitalWrite(ledPin, LOW);` turns the LED off.

1.2.5 Test your Circuit

The system is working correctly if:

1. The LED turns on when the pushbutton is pressed
2. The LED turns off when the pushbutton is released

You can test this by pressing and holding the button — the LED should remain lit for as long as the button is held down. When you release the button, the LED should turn off immediately. If this behavior does not occur, double-check the wiring of your button, LED, resistor, and the code logic. Ensure that the button pin is properly pulled down using a resistor (typically 10k Ω as discussed in Section 1.2.1) to avoid a floating input.

1.2.6 Aside – Multimeters

A multimeter is a valuable tool for debugging and exploring how electricity flows in a circuit. Whether you're using a handheld multimeter or a bench/tabletop multimeter, the basic functions are the same. Here's how to measure resistance, voltage, and current in this project:

1.2.6a - Measuring Resistance

1. Power off your Arduino.
2. Disconnect one leg of the resistor from the breadboard to avoid interference from the circuit.
3. Set the dial to the Ω (ohm) symbol.
4. Touch one probe to each side of the resistor (see Figure 10)
5. You should see a reading close to 220 Ω .

Tip: Handheld multimeters may require manual range setting, while tabletop multimeters often auto-range and may give more precise decimal values.

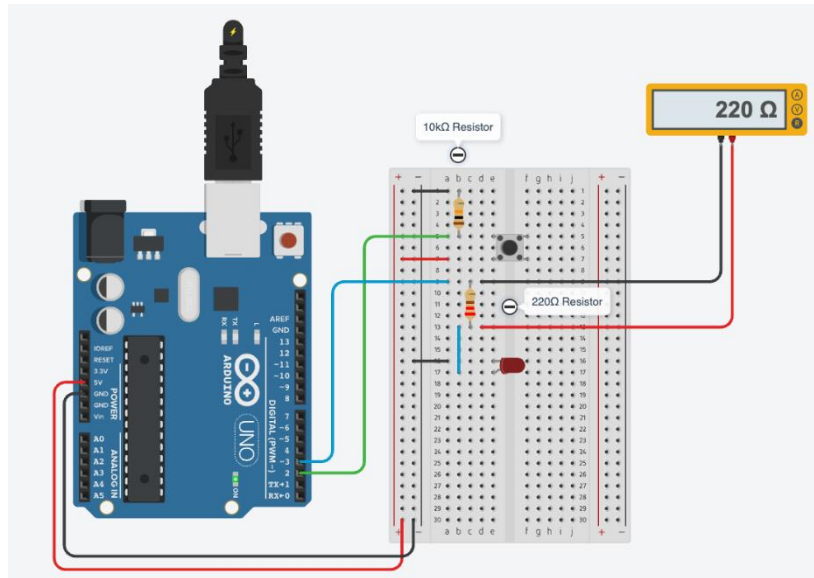


Figure 10 - Measuring Resistance

1.2.6b - Measuring Voltage

1. Keep the circuit powered by uploading the Arduino code and pressing the button.
2. Set the dial to DC Voltage (V_{DC}).
3. Place the black (COM) probe on GND.
4. Place the red probe on the LED's anode (the longer leg).
5. When the button is pressed, you'll see:
 - a. ~2V across the LED (its voltage drop)

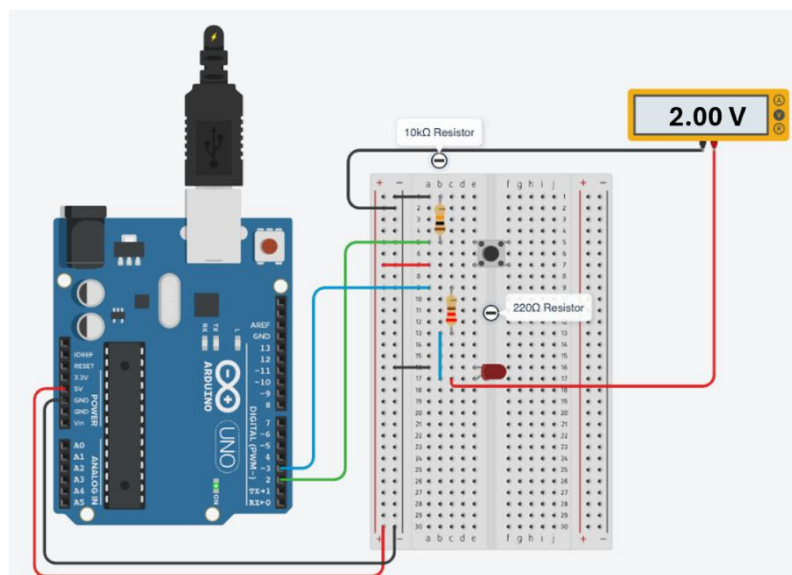


Figure 11- Measuring Voltage

1.2.6c - Measuring Current

Important: You must connect the multimeter in series, not parallel!

1. Turn off power.
2. Locate the wire that connects digital pin 3 to the 220-ohm resistor.
 - a. Remove that wire from pin 3 (leave it connected to the resistor)
3. Set the dial to DC current (mA or A).
4. Move the red probe to the “A” or “mA” port (check your meter).
5. Connect the black probe to digital pin 3, and the red to the resistor leg (or the now free end of the wire going to the resistor).
6. Power on and press the button — you should read about 10 – 15 mA.

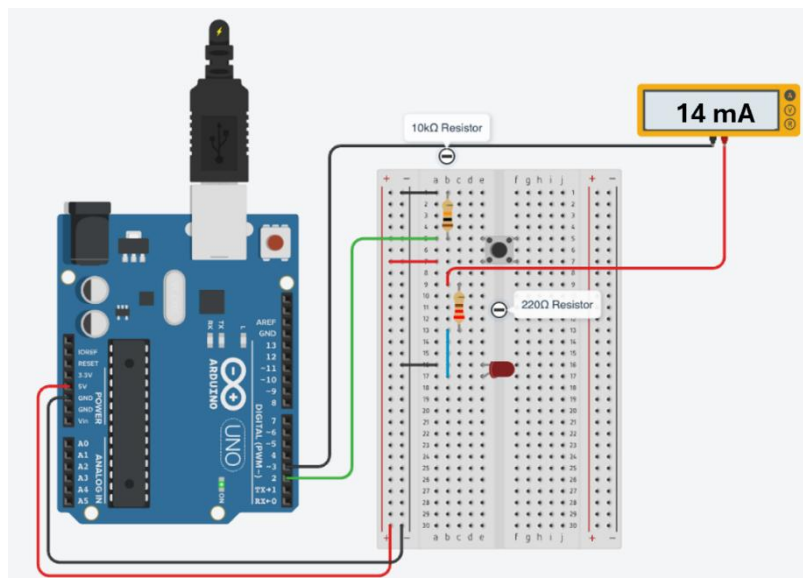


Figure 12 – Measuring Current

Multimeters are essential tools for electronics — the earlier you get comfortable using one, the more confident you’ll be in building and troubleshooting your circuits!

1.3 Challenge 1

Create a system that uses two pushbuttons:

1. Button 1 turns the LED on
2. Button 2 turns the LED off

The LED should stay on after the "On" button is released and only turn off when the "Off" button is pressed. Likewise, the LED should stay off after the "Off" button is released, until the "On" button is pressed again.

This behavior is known as a latching system — the LED "remembers" its state.

2 Activity – Analog Inputs and UART

Analog signals are everywhere in the real world, like temperature, light, or sound. Microcontrollers cannot process these signals directly without an Analog-to-Digital Converter (ADC). Learning to work with ADCs is crucial because sensors that detect analog signals are key in robotics, automation, and even biomedical applications. Understanding how to read analog signals and convert them into data that a microcontroller can use opens possibilities for building complex, responsive systems.

2.1 Background

2.1.1 Analog Signals

Digital signals were defined as signals that can take on 2 different states or voltage levels in section 1. Analog signals, in contrast, can take on any voltage between its minimum and maximum.

2.1.2 Measuring Analog Signals using Microcontrollers – ADCs

Microcontrollers are digital devices, which makes it somewhat complicated to read analog signals. Specialized devices called ADCs (Analog-to-Digital Converters) are used to convert analog voltages into digital values that microcontrollers can understand. The Arduino Uno has built-in 10-bit ADCs on its analog pins (A0–A5), making this process straightforward. This means each analog input is represented by one of 2^{10} or 1024 distinct values where 0 represents 0 V and 1023 represents 5 V.

2.1.3 Potentiometers

A potentiometer acts as a variable voltage divider. The outer pins are connected to a voltage supply and ground, while the middle pin outputs a voltage between these two values, depending on the position of the dial. The output voltage varies linearly with the dial's angle, ranging from 0 V (ground) to the supplied voltage (5 V in this case).

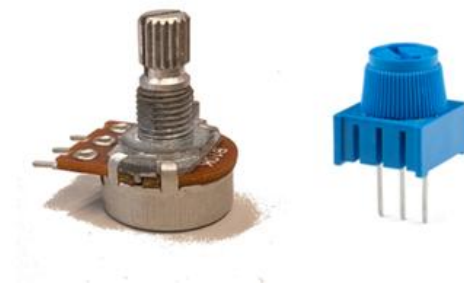


Figure 13 - Potentiometer

2.1.4 UART and Serial Communication

UART is an important protocol for data exchange between devices. Many embedded systems require communication between the microcontroller and a computer or another microcontroller. Mastering UART is necessary for creating systems that can share data, like sensors sending information to a central processor or logging data to a computer.

The Arduino Uno has two dedicated UART pins:

- D0 – RX (Receive)
- D1 – TX (Transmit)

It also has a built-in USB-to-Serial chip. When you connect your Arduino to your computer via USB and use the **Serial.begin()** and **Serial.println()** functions, you're communicating through these UART pins. There's no need to connect anything to the physical pins.

2.2 Task

In this task, you will connect the 5V pin to a potentiometer, read its voltage using an analog input pin, and display the value in the Serial Monitor.

2.2.1 Wire your Circuit

Refer to Figure 14 for the circuit's wiring.

1. Use the breadboard's power and ground rails to connect the outer pins of the potentiometer
2. Connect the power and ground rails to 5V and GND on the Arduino.
3. Connect the middle (wiper) pin of the potentiometer to analog pin A0.

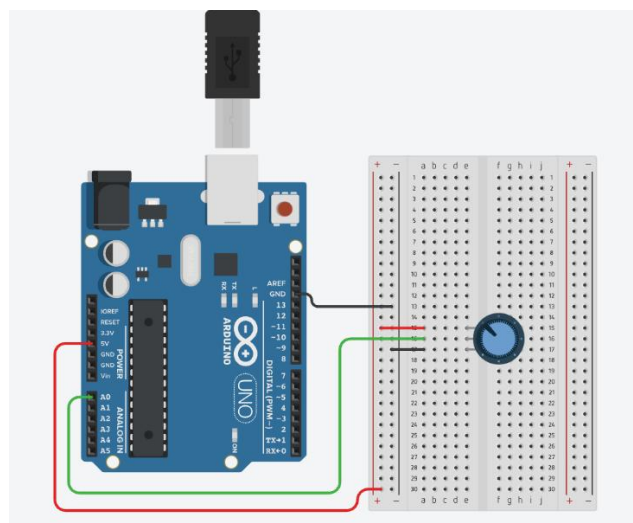


Figure 14 - Layout Diagram for Potentiometer

2.2.2. Add your Code

Once again, the code is provided for you. There is no special configuration is required - just open the Arduino IDE and paste this code into a new sketch/project:

```
-----  
int potPin = A0;           // Potentiometer connected to analog pin A0  
int adcValue = 0;          // Variable to store the ADC reading  
  
void setup() {  
    Serial.begin(9600);     // Start serial communication at 9600 baud  
}  
  
void loop() {  
    adcValue = analogRead(potPin); // Read analog voltage (0-1023)  
    Serial.println(adcValue);      // Print the value to Serial Monitor  
    delay(100);                   // Small delay to make output readable  
}  
-----
```

This code reads a variable voltage between 0V and 5V using analog input pin A0.

- **analogRead(potPin)** reads the voltage on analog pin A0 and converts it to a 10-bit digital value (0 -1023)
- **Serial.begin(9600)** starts serial communication so you can view the readings in the Serial Monitor
- **Serial.println(adcValue)** prints the result to your computer

2.3 Challenge 2

Use a potentiometer to control the frequency of a blinking LED. See the Potentiometers section for how to use the potentiometer. The outer pins are connected to a voltage supply and ground, while the middle pin outputs a voltage between these two values, depending on the position of the dial.

3. PWM and Oscilloscopes

PWM (Pulse Width Modulation) is a versatile technique used for controlling devices like motors and LEDs with variable brightness. It's widely used in power control applications, signal modulation, and communication systems. Learning how to generate PWM signals and analyze them with an oscilloscope will help you develop a

clear understanding of how to create efficient control systems in real-world applications. These skills apply directly to automotive systems, robotics, and any system that requires precision control.

3.1. Background

3.1.1. PWM

PWM (Pulse Width Modulation) is a technique that is used to approximate analog signals using digital signals. Figure 15 demonstrates the concept of duty cycle, which is the percentage of time that the signal is in the “High” state for. Figure 15 highlights a duty cycle’s amplitude, period and frequency. Amplitude is the voltage of the “High” state, or the maximum voltage that the signal reaches. The period is the time for one full cycle to occur (time from rising edge to rising edge). The frequency is 1 divided by the period.

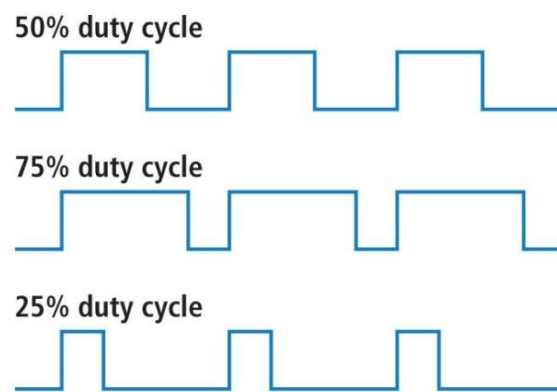


Figure 15 –PWM Duty Cycles

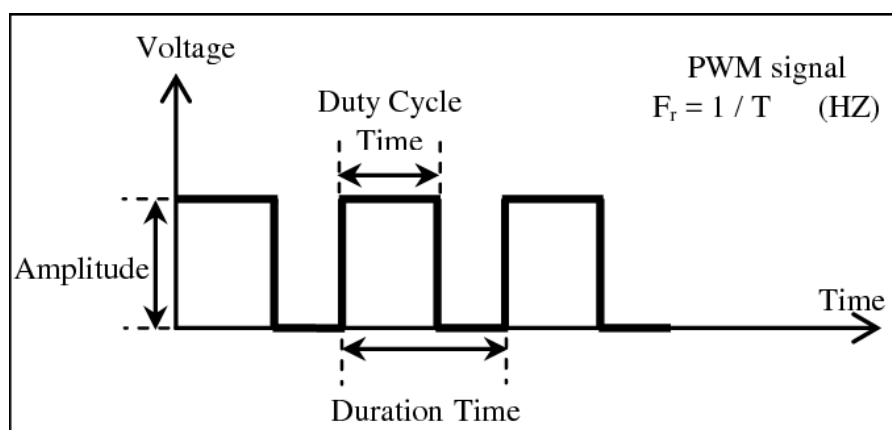


Figure 16 - PWM Frequency and Amplitude

The recommended PWM pins for the Arduino Uno are digital pins 3, 5, 6, 9, 10, and 11. You can identify these pins on the Pinout map because they’re marked with a tilde (~).

3.1.2. Oscilloscopes

Oscilloscopes are electronic devices that are used to view the voltage of a signal over time. Cursors can be used to measure the time and voltage of signals. They are very useful for hardware debugging since they give a visual of signals which can be compared with expected results.

3.2. Task

In this task, you will generate a PWM signal on the Arduino board and view the square waves on an oscilloscope. Your potentiometer will control the percentage of time the duty cycle is high.

3.3.1. Wire your Circuit

There's not much to change in your circuit. You'll connect the oscilloscope probes to the Arduino as shown in Figure 17. (Yellow to D9, Black to GND).

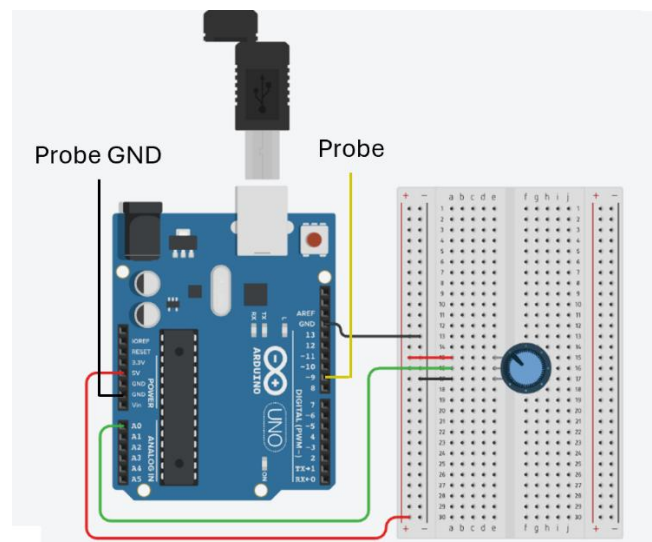


Figure 17– PWM connections

3.3.2. Add your Code

Once again, the code is provided for you. There is no special configuration is required - just open the Arduino IDE and paste this code into a new sketch/project:

```
-----  
  
int ledPin = 9;          // LED connected to digital pin 9  
  
int analogPin = A0;     // potentiometer connected to analog pin A0  
  
int val = 0;            // variable to store the read value  
  
void setup() {  
    pinMode(ledPin, OUTPUT); // sets the pin as output
```

```
}
```

```
void loop() {  
    val = analogRead(analogPin); // read the input pin  
    analogWrite(ledPin, val / 4);  
    // analogRead values go from 0 to 1023, analogWrite values from 0 to 255  
}
```

This code reads voltage between 0V and 5V from the potentiometer and outputs a PWM signal that you'll view on the oscilloscope.

- **analogRead()** reads the voltage on the specified analog pin and converts it to a 10-bit digital value (0 -1023)
- **analogWrite ()** divides the digital equivalent of the analog value by 4, outputting an 8-bit value (0 – 255) to the specified digital PWM output pin

3.3.3. View the PWM signal

Once your code is running, **click** the “**Auto**” button on the oscilloscope. This will automatically configure the voltage and timing scale to view the signal clearly. If this does not work, try setting the voltage scale to 1V/div and the timing scale to 100 microseconds. The PWM signal should be a square wave like what's shown in Figure 18. Note that the duty cycle is 50%, the period is 1 ms, and the frequency is 1000 Hz.

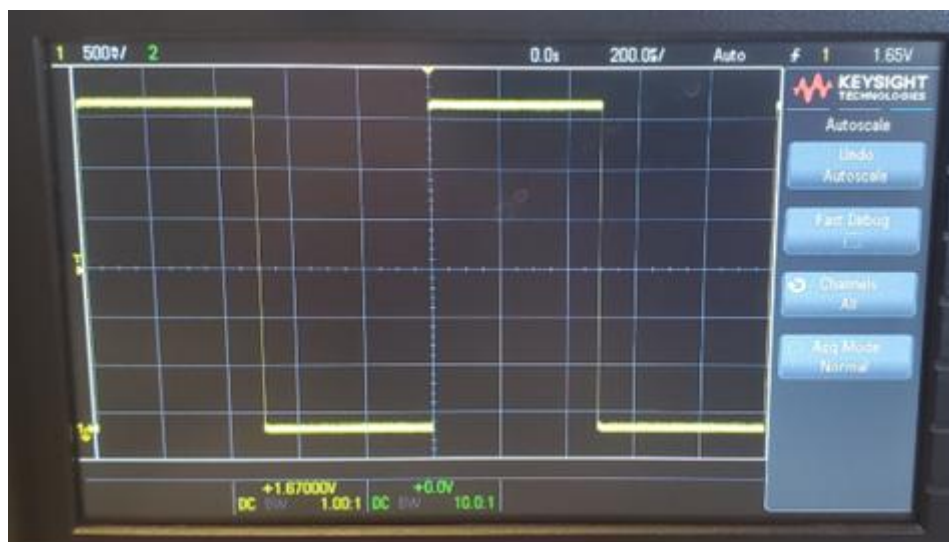


Figure 18 - PWM Results

3.3. Challenge 3

Add two LEDs to your circuit. One can be connected to D9 where you observed the waveforms, and the other to one of the Arduino's other PWM output pins (see section 3.1.1). Don't forget the current-limiting resistors.

Your challenge is to vary the intensity of the two LEDs such that one is driven at the level specified by the potentiometer (as the code does right now) and the other by its complement. For example, if calculated value controlling the first LED is 150, then the other is $(255-150)$ or 115. If the calculated value is 25, then the value for the 2nd LED is $255-25$ or 230.

Modify the code from section 3.3.2 to create this very simple light show. As you vary the PWM signal, the two LEDs fade or brighten accordingly.

4. Related Resources

The following resources may be useful as you start to use an Arduino for other projects:

- [Resistor Color Code Calculator](#)
- [Hints about Wiring for Your Electronics Projects](#)