Project 12
PWM Control

About This Project

In Arduino PWM Tutorial, you are going to learn about what PWM is and how you can get the PWM output from the digital pins of Arduino. First, we will control the brightness of LED through code and then we will control it manually by adding the potentiometer.

What is PWM

PWM stands for Pulse Width Modulation and it is a technique used in controlling the brightness of LED, speed control of DC motor, controlling a servo motor or where you have to get analog output with digital means.

The Arduino digital pins either gives us 5V (when turned HIGH) or 0V (when turned LOW) and the output is a square wave signal. So, if we want to dim a LED, we cannot get the voltage between 0 and 5V from the digital pin but we can change the ON and OFF time of the signal. If we will change the ON and OFF time fast enough then the brightness of the led will be changed.

Before going further, let's discuss some terms associated with PWM.

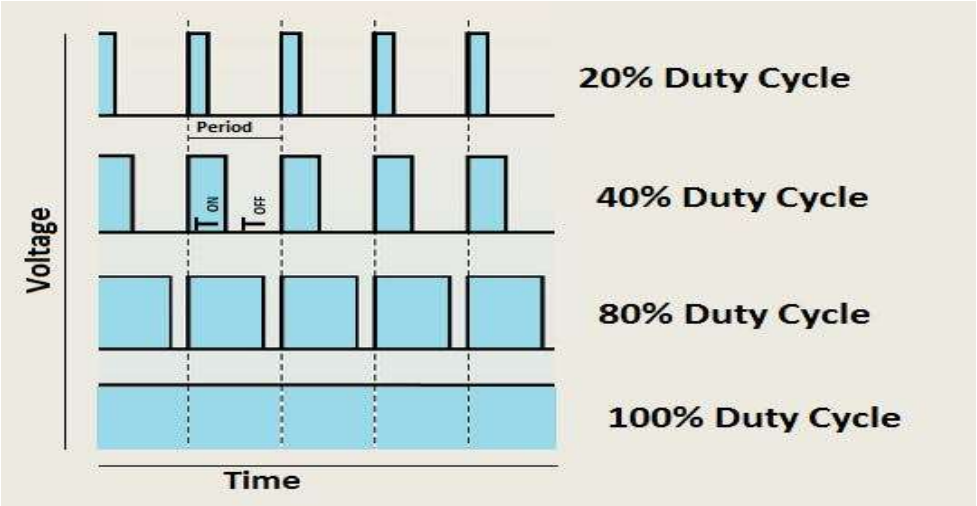
TON (On Time): It is the time when the signal is high.

TOFF (Off Time): It is the time when the signal is low.

Period: It is the sum of on time and off time.

Duty Cycle: It is the percentage of time when the signal was high during the time of period.

So, at 50% duty cycle and 1Hz frequency, the led will be high for half a second and will be low for the other half second. If we increase the frequency to 50Hz (50 times ON and OFF per second), then the led will be seen glowing at half brightness by the human eye.



Arduino and PWM

The Arduino IDE has a built in function “analogWrite()” which can be used to generate a PWM signal. The frequency of this generated signal for most pins will be about 490Hz and we can give the value from 0-255 using this function.

analogWrite(0) means a signal of 0% duty cycle.

analogWrite(127) means a signal of 50% duty cycle.

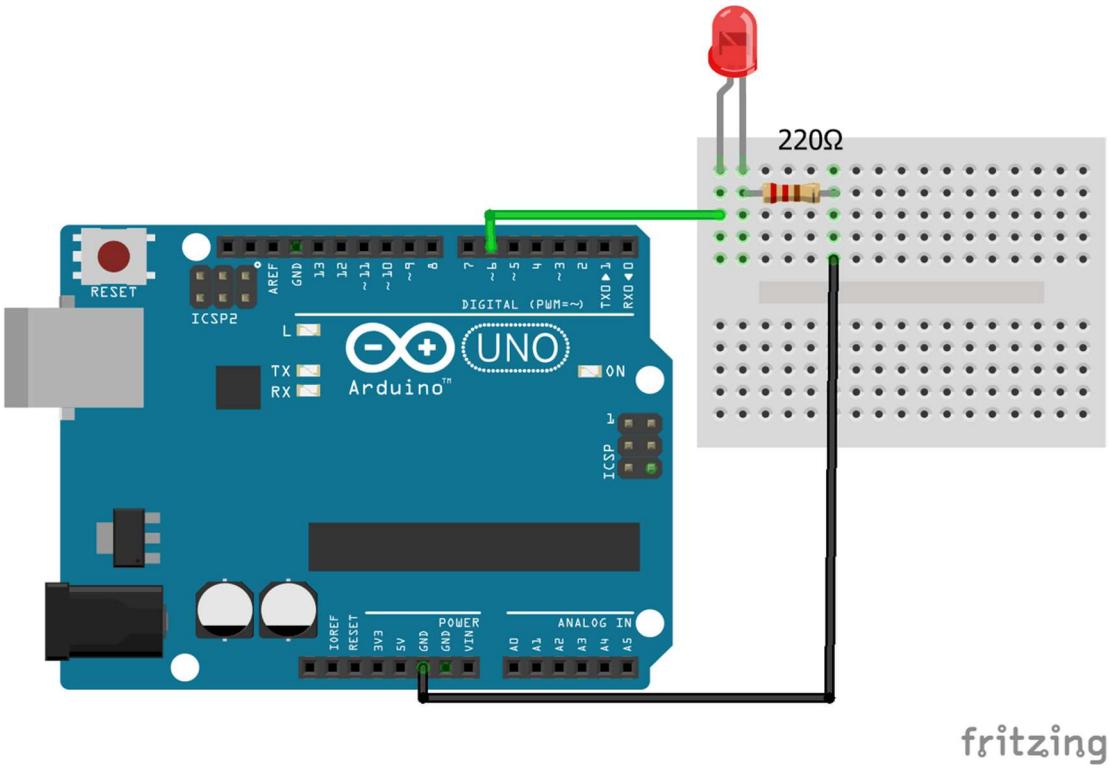
analogWrite(255) means a signal of 100% duty cycle.

On Arduino Uno, the PWM pins are 3, 5, 6, 9, 10 and 11. The frequency of PWM signal on pins 5 and 6 will be about 980Hz and on other pins will be 490Hz. The PWM pins are labeled with ~ sign.

Controlling Brightness of LED through Code

Firstly, make the connections as described below.

Connect the positive leg of LED which is the longer leg to the digital pin 6 of Arduino. Then connect the 220 ohm resistor to the negative leg of LED and connect the other end of resistor to the ground pin of Arduino.



Now let's write a code to change the brightness of the LED using PWM.

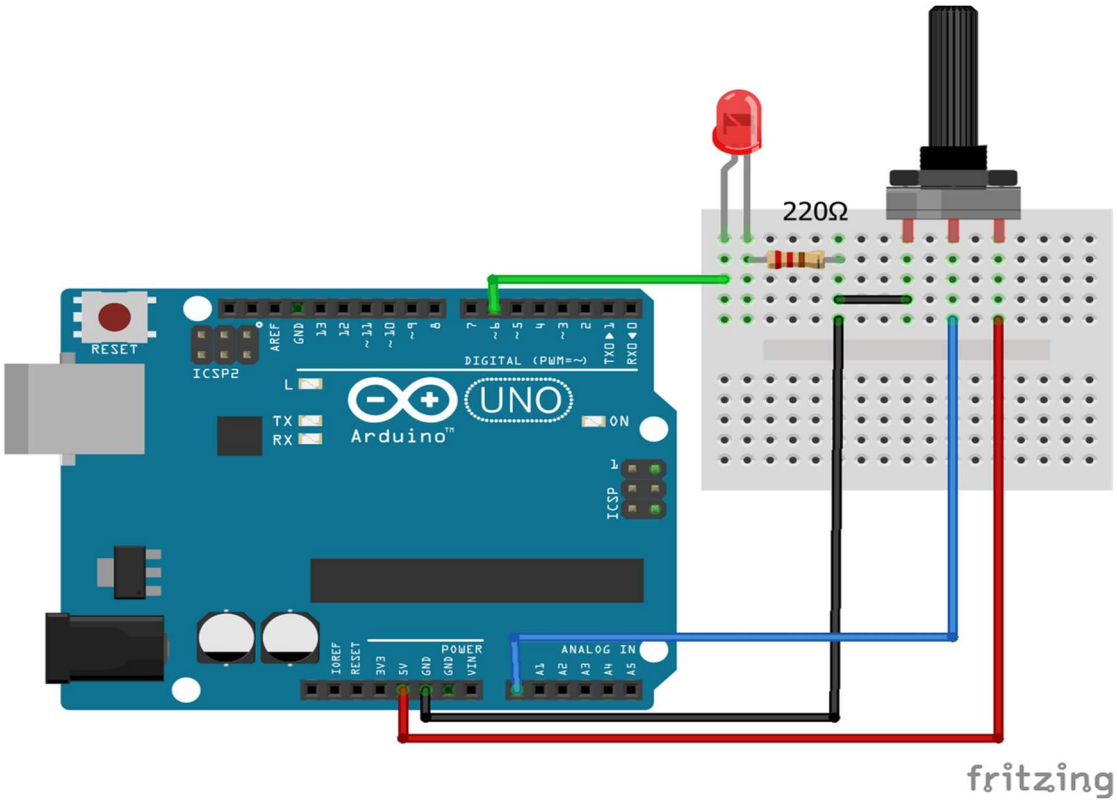
Arduino Code

Upload the code in the Arduino IDE and the LED will start to fade.

```
//Initializing LED Pin
int led_pin = 6;
void setup() {
  //Declaring LED pin as output
  pinMode(led_pin, OUTPUT);
}
void loop() {
  //Fading the LED
  for(int i=0; i<255; i++){
    analogWrite(led_pin, i);
    delay(5);
  }
  for(int i=255; i>0; i--){
    analogWrite(led_pin, i);
    delay(5);
  }
}
```

Arduino Code to manually control the Brightness of LED

In the previous connections, add the 10k ohm potentiometer and connect the two ends of potentiometer to 5V and GND of Arduino and then connect the center of potentiometer to the A0 pin of Arduino.




```
int led_pin = 6;
int pot_pin = A0;
int output;
int led_value;
void setup() {
  pinMode(led_pin, OUTPUT);
}
void loop() {
  //Reading from potentiometer
  output = analogRead(pot_pin);
  //Mapping the Values between 0 to 255 because we can give output
  //from 0 -255 using the analogwrite funtion
  led_value = map(output, 0, 1023, 0, 255);
  analogWrite(led_pin, led_value);
  delay(1);
}
```

Analog to Digital Conversion

The Analog World

Microcontrollers are capable of detecting binary signals: is the button pressed or not? These are digital signals. When a microcontroller is powered from five volts, it understands zero volts (0V) as a binary 0 and five volts (5V) as a binary 1. The world however is not so simple and likes to use shades of gray. What if the signal is 2.72V? Is that a zero or a one? We often need to measure signals that vary; these are called analog signals. A 5V analog sensor may output 0.01V or 4.99V or anything in between. Luckily, nearly all microcontrollers have a device built into them that allows us to convert these voltages into values that we can use in a program to make a decision.

What is the ADC?

An Analog to Digital Converter (ADC) is a very useful feature that converts an analog voltage on a pin to a digital number. By converting from the analog world to the digital world, we can begin to use electronics to interface to the analog world around us.

Not every pin on a microcontroller has the ability to do analog to digital conversions. On the Arduino board, these pins have an ‘A’ in front of their label (A0 through A5) to indicate these pins can read analog voltages.

ADCs can vary greatly between microcontroller. The ADC on the Arduino is a 10-bit ADC meaning it has the ability to detect 1,024 (2^10) discrete analog levels. Some

microcontrollers have 8-bit ADCs (2^8 = 256 discrete levels) and some have 16-bit ADCs (2^16 = 65,536 discrete levels).

The way an ADC works is fairly complex. There are a few different ways to achieve this feat (see Wikipedia for a list), but one of the most common technique uses the analog voltage to charge up an internal capacitor and then measure the time it takes to discharge across an internal resistor. The microcontroller monitors the number of clock cycles that pass before the capacitor is discharged. This number of cycles is the number that is returned once the ADC is complete.

Relating ADC Value to Voltage

The ADC reports a ratiometric value. This means that the ADC assumes 5V is 1023 and anything less than 5V will be a ratio between 5V and 1023.

$$\frac{Resolution\ of\ the\ ADC}{System\ Voltage} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

Analog to digital conversions are dependent on the system voltage. Because we predominantly use the 10-bit ADC of the Arduino on a 5V system, we can simplify this equation slightly:

$$\frac{1023}{5} = \frac{ADC\ Reading}{Analog\ Voltage\ Measured}$$

If your system is 3.3V, you simply change 5V out with 3.3V in the equation. If your system is 3.3V and your ADC is reporting 512, what is the voltage measured? It is approximately 1.65V.

If the analog voltage is 2.12V what will the ADC report as a value?

$$\frac{1023}{5.00V} = \frac{x}{2.12V}$$

Rearrange things a bit and we get:

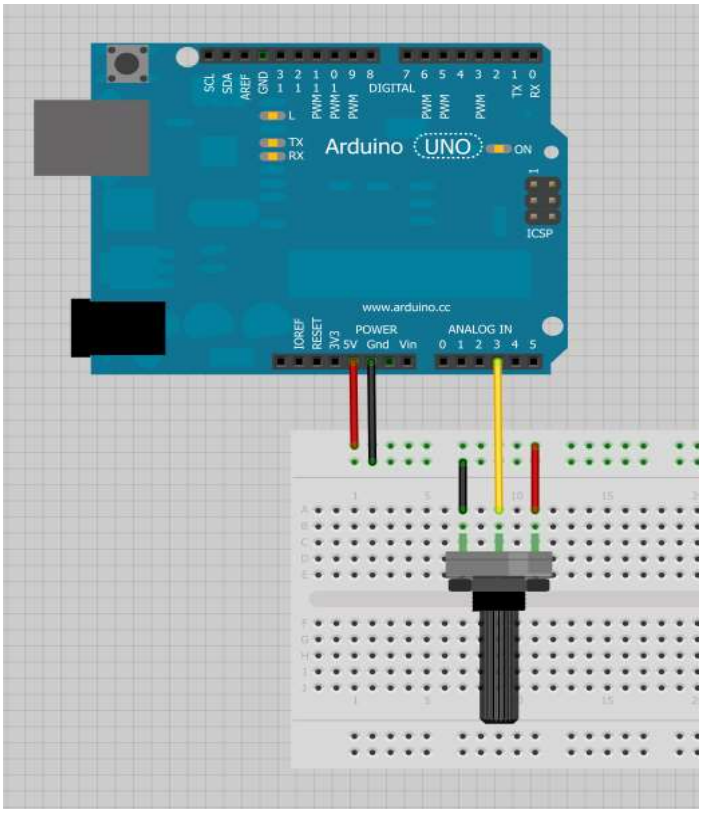
$$\frac{1023}{5.00V} * 2.12V = x$$

$$x = 434$$

Ahah! The ADC should report 434.

Arduino ADC Example

To show this in the real world let's use the Arduino to detect an analog voltage. Use a trimpot, or light sensor, or simple voltage divider to create a voltage. Let's setup a simple trimpot circuit for this example:



To start, we need to define the pin as an input. To match the circuit diagram we will use A3:

```
pinMode(A3, INPUT);
```

and then do the analog to digital version by using the analogRead() command:

```
int x = analogRead(A3); //Reads the analog value on pin A3 into x
```

The value that is returned and stored in x will be a value from 0 to 1023. The Arduino has a 10-bit ADC (2^10 = 1024). We store this value into an int because x is bigger (10 bits) than what a byte can hold (8 bits).

Let's print this value to watch it as it changes:

```
COPY CODESerial.print("Analog value: ");
Serial.println(x);
```

As we change the analog value, x should also change. For example, if x is reported to be 334, and we're using the Arduino at 5V, what is the actual voltage? Pull out your digital multimeter and check the actual voltage. It should be approximately 1.63V. Congratulations! You have just created your own digital multimeter with an Arduino!

Hooking Things Up Backward

What happens if you connect an analog sensor to a regular (digital) pin? Nothing bad will happen. You just won't be able to do an analogRead successfully:

```
COPY CODEint x = analogRead(8); //Try to read the analog value on digital pin 8 - this doesn't
work!
```

This will compile but x will be filled with a nonsensical value.

What happens if I connect a digital sensor to an analog pin? Again, you will not break anything. If you do an analog-to-digital conversion on a button, you will most likely see ADC values very close to 1023 (or 5V which is binary 1) or very close to 0 (or 0V which is binary 0).