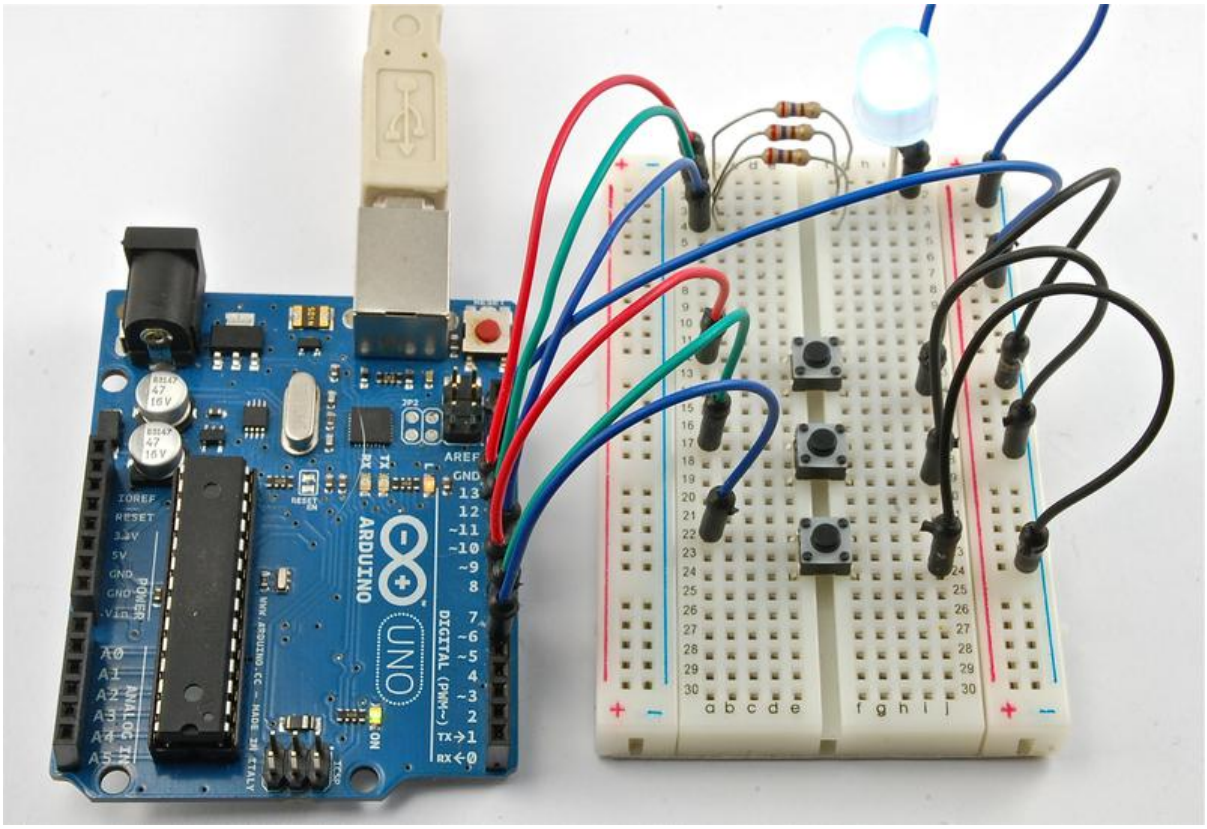
Arduino Lesson 7. Make an RGB LED Fader

Created by Simon Monk



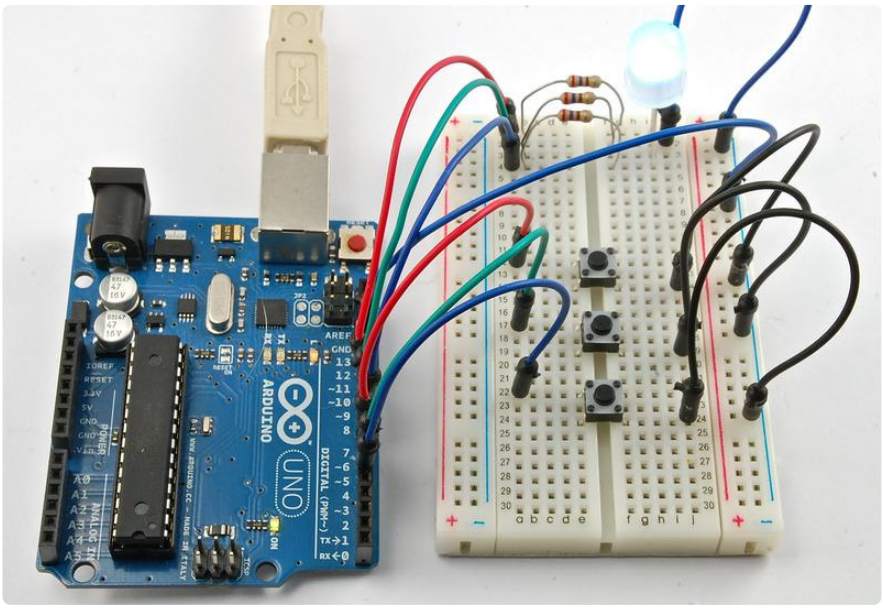
https://learn.adafruit.com/adafruit-arduino-lesson-7-make-an-rgb-led-fader

Last updated on 2023-08-29 02:12:15 PM EDT

Table of Contents

Overview

In this lesson, you will be combining some of the things that you learnt in earlier lessons to use three push switches (from lesson 6) to control the color of the RGB LED that you used in lesson 3 in order to make a Color Organ!



Parts

To carry out the experiment described in this lesson, you will need the following parts.

Part
Qty

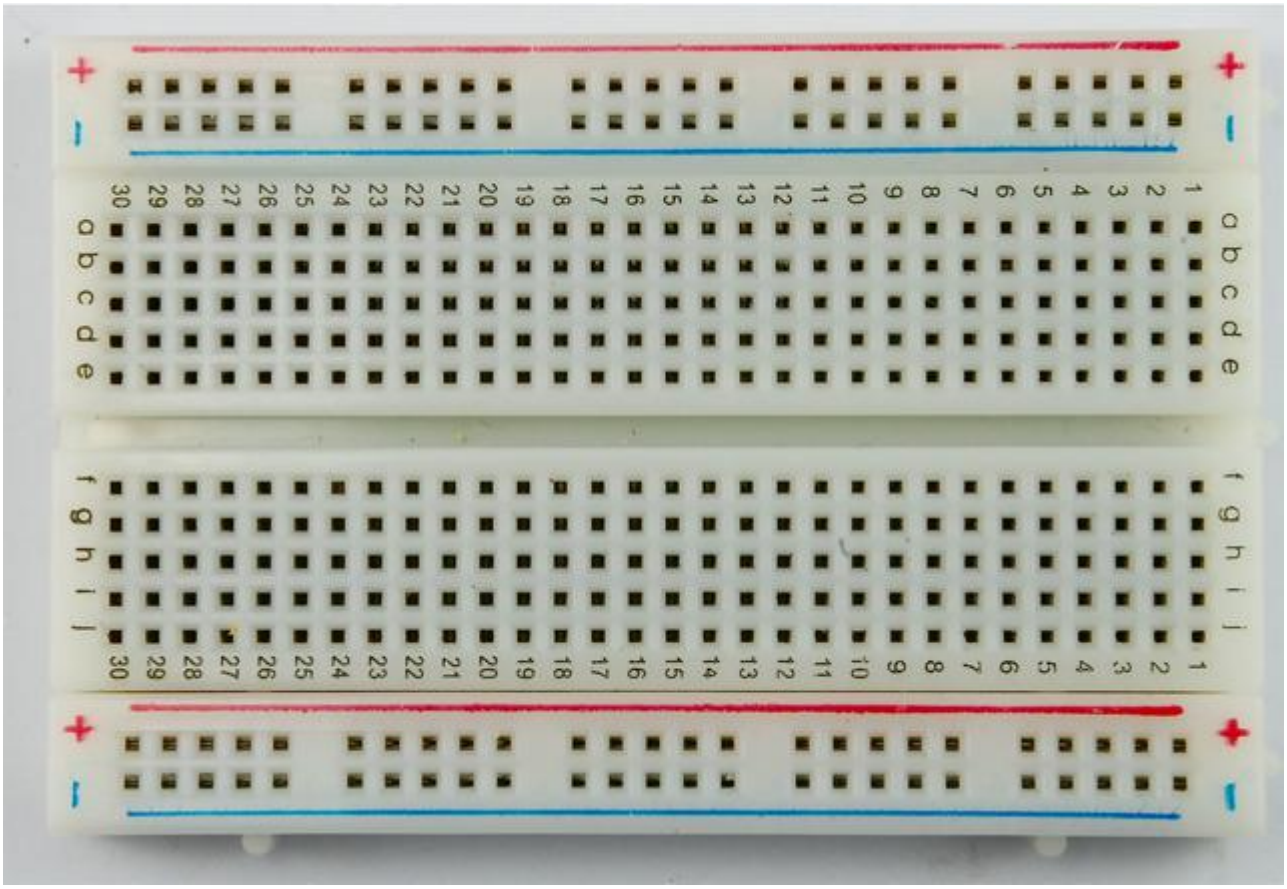
10mm Common Cathode RGB LED
1

270 Ω Resistor (red, purple, brown stripes)

3

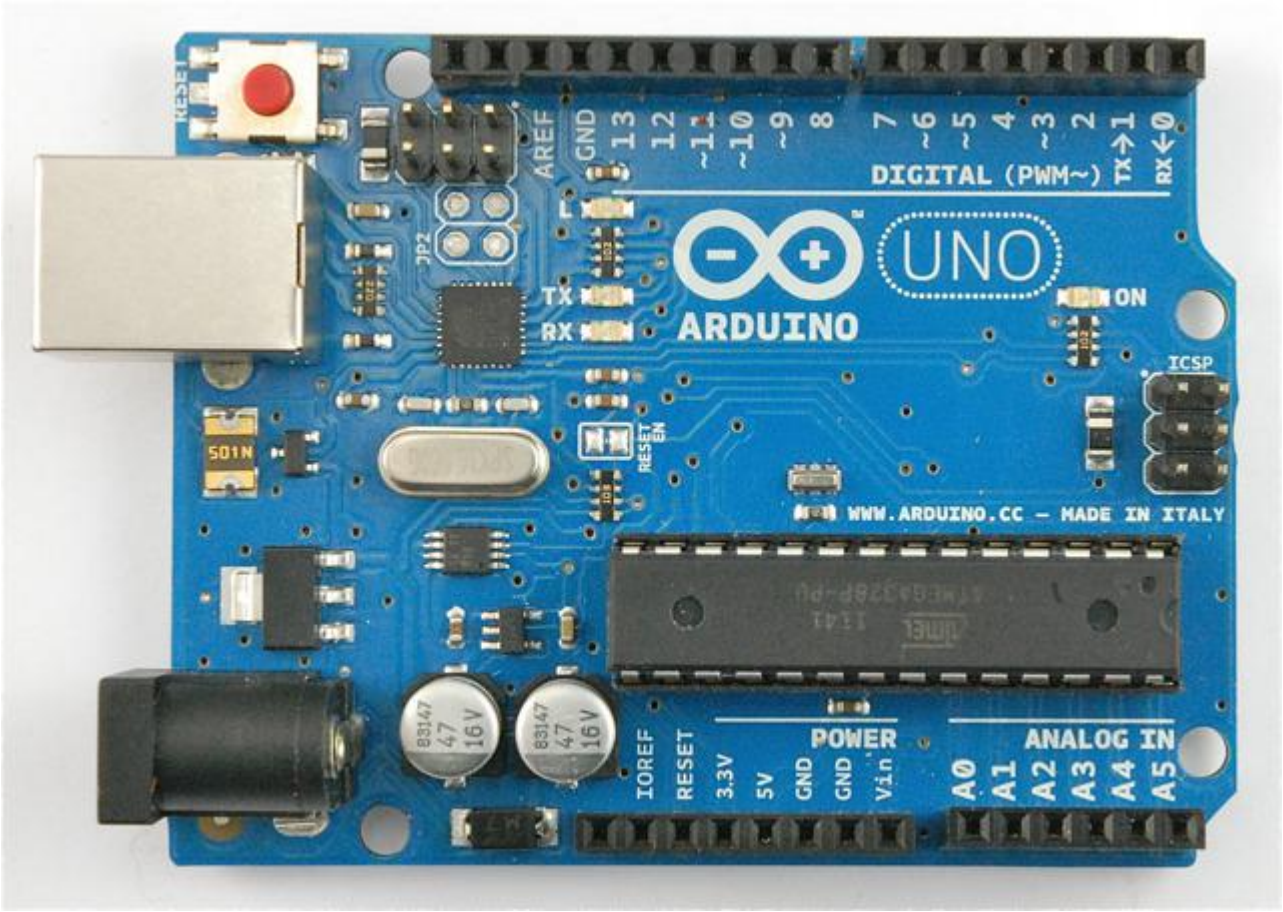
Tactile push switch

3



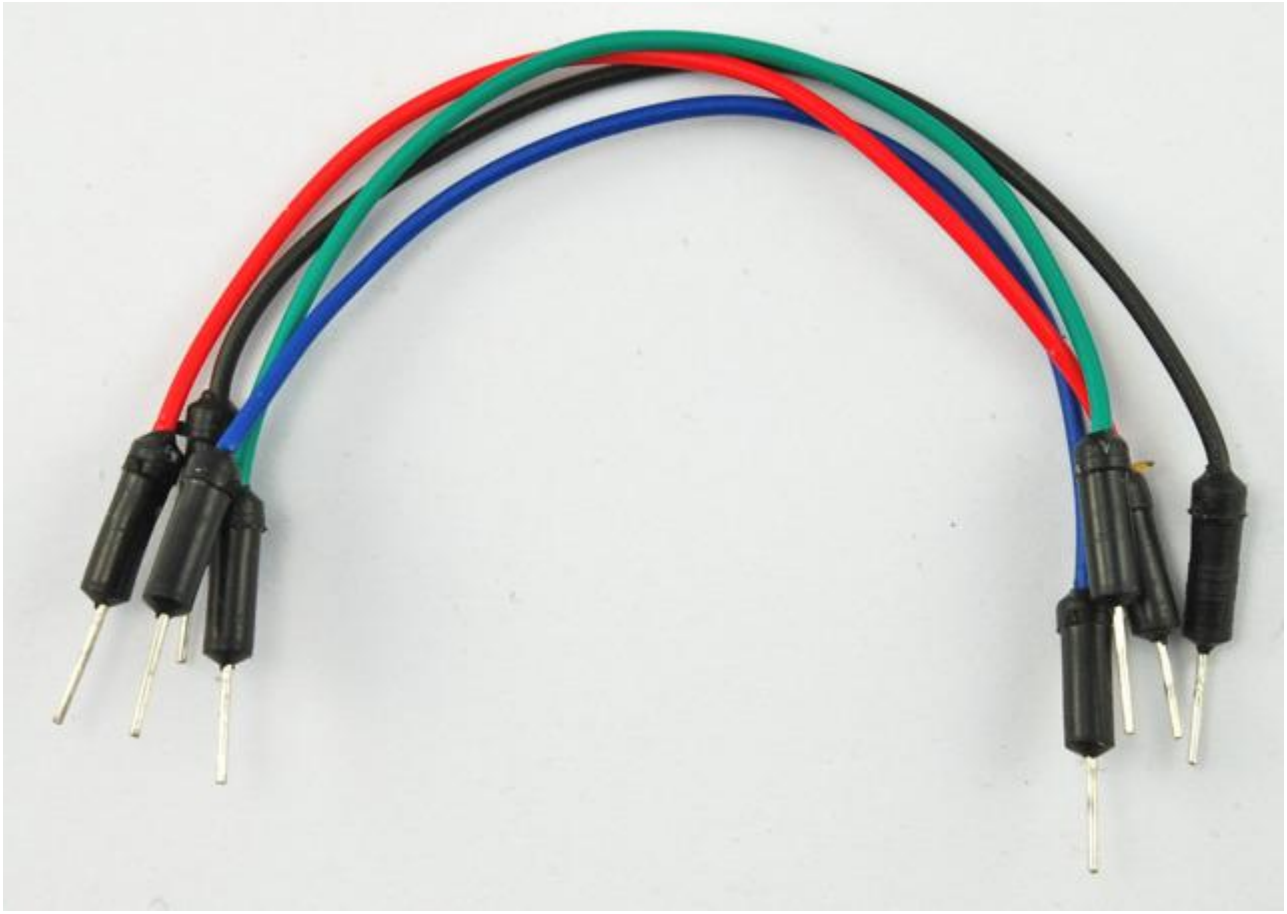
Half-size Breadboard

1



Arduino Uno R3

1



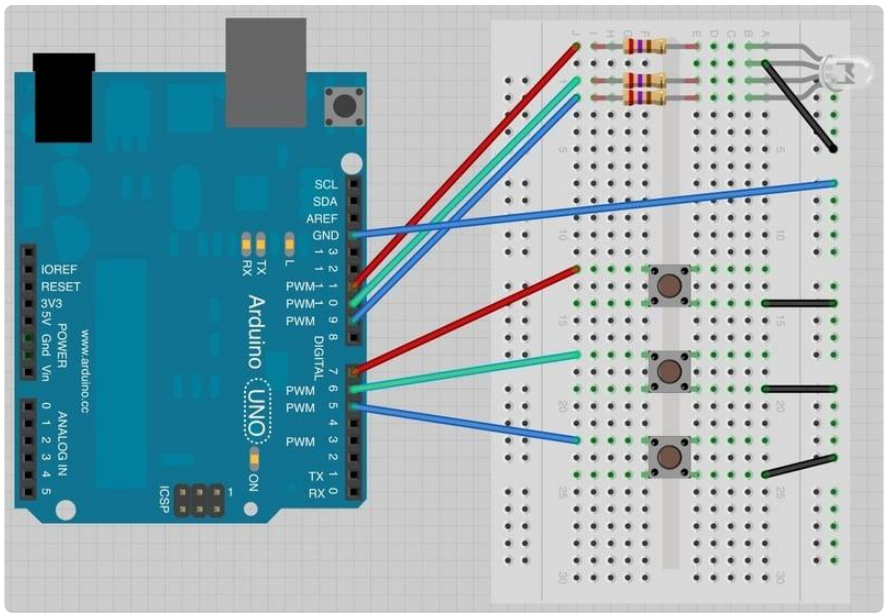
Jumper wire pack

1

Breadboard Layout

The breadboard layout is shown below. The RGB LED should have the longest lead in row 2 of the breadboard. This will be connected to GND.

This wiring diagram assumes you have a common cathode LED - if you have a common anode LED, connect the longest pin to +5V instead of ground. Note that with a common anode display the LED color cycling will be reversed.



Arduino Code

Load the following sketch onto your Arduino board.

To start with, all the LEDs will be off. If you press and hold one of the buttons, then the LED will gradually get brighter. The color will be red for the top button, green for the middle one and blue for the bottom button.

When you have got enough of one color, try pressing another button and see how they mix together.

If you want to start over, then press the reset button on the Arduino. This is the red button, next to the USB connector.

```
/*
Adafruit Arduino - Lesson 7. RGB Fader
*/

int redLEDPin = 11;
int greenLEDPin = 10;
int blueLEDPin = 9;

int redSwitchPin = 7;
int greenSwitchPin = 6;
int blueSwitchPin = 5;

int red = 0;
int blue = 0;
int green = 0;

void setup()
{
  pinMode(redLEDPin, OUTPUT);
  pinMode(greenLEDPin, OUTPUT);
  pinMode(blueLEDPin, OUTPUT);
```

```
  pinMode(redSwitchPin, INPUT_PULLUP);
  pinMode(greenSwitchPin, INPUT_PULLUP);
  pinMode(blueSwitchPin, INPUT_PULLUP);
}

void loop()
{
  if (digitalRead(redSwitchPin) == LOW)
  {
    red ++;
    if (red &gt; 255) red = 0;
  }
  if (digitalRead(greenSwitchPin) == LOW)
  {
    green ++;
    if (green &gt; 255) green = 0;
  }
  if (digitalRead(blueSwitchPin) == LOW)
  {
    blue ++;
    if (blue &gt; 255) blue = 0;
  }
  analogWrite(redLEDPin, red);
  analogWrite(greenLEDPin, green);
  analogWrite(blueLEDPin, blue);
  delay(10);
}
```

The sketch is similar to that of lesson 3. Again, we have three output pins to control the LED. These are PWM pins so that we can control how much power goes to each color of the LED.

There are three other pins needed, one for each of the buttons and these are configured in 'setup' to be inputs that are pulled up to HIGH, unless the button is pressed, which will cause them to become LOW.

After the pin definition, there is another little group of variables called 'red', 'green' and 'blue'.

```
int red = 0;
int blue = 0;
int green = 0;
```

These variables are used to hold the current values for the intensity of each of the LED channels. So, if 'red' is 0 then the red part of the LED will be off. However, if it is 255 then it will be at maximum brightness.

The 'loop' function is in two parts. The first part checks the buttons and makes any necessary changes to 'red', 'green' or 'blue' depending on the button. Each button works in the same way, just for a different color. The section for checking the red button is like this:

```
  if (digitalRead(redSwitchPin) == LOW)
  {
```

```
    red ++;
    if (red &gt; 255) red = 0;
  }
```

If, when we carry out a 'digitalRead' on the red switch pin, we find it is LOW, then that means the button is pressed, so we add 1 to 'red'. The command ++ adds one to a variable.

However, we need to be careful here, because the maximum value that you can use for PWM is 255. So on the next line, we check to see if we have gone over the limit, and if we have then red is set back to 0, so we start over.

The second part of the 'loop' function carries out the 'analogWrite's to each of the LED colors.

```
  analogWrite(redLEDPin, red);
  analogWrite(greenLEDPin, green);
  analogWrite(blueLEDPin, blue);
```

Finally there is a short delay at the end of the loop, that slows down the color changing to a manageable speed.

Other Things to Do

Try removing the delay at the end of 'loop'. You can do this by 'commenting out' the delay line by putting // at the start of the line like this:

```
  analogWrite(blueLEDPin, blue);
  // delay(10);
}
```

This is a useful technique, because when you find out that you need to put the line back in again, you can do so, just by taking the // away again.

Without the delay, you will find that basically, whenever you release the button, you will be left with a random level of brightness for that color. What is happening is that the it is cycling through the brightness levels so fast, that you have no chance of actually controlling it.

Another experiment you could make would be to change the function of the three buttons so that they control the LED as if it was a traffic signal.

Try to make it so that the top button turns the LED fully green, the middle button Orange and the bottom button red.

Click Here for the Next Lesson

About the Author

Simon Monk is author of a number of books relating to Open Source Hardware. The following books written by Simon are available from Adafruit: Programming Arduino (http://adafru.it/1019), 30 Arduino Projects for the Evil Genius (http://adafru.it/868) and Programming the Raspberry Pi ().