adafruit learning system

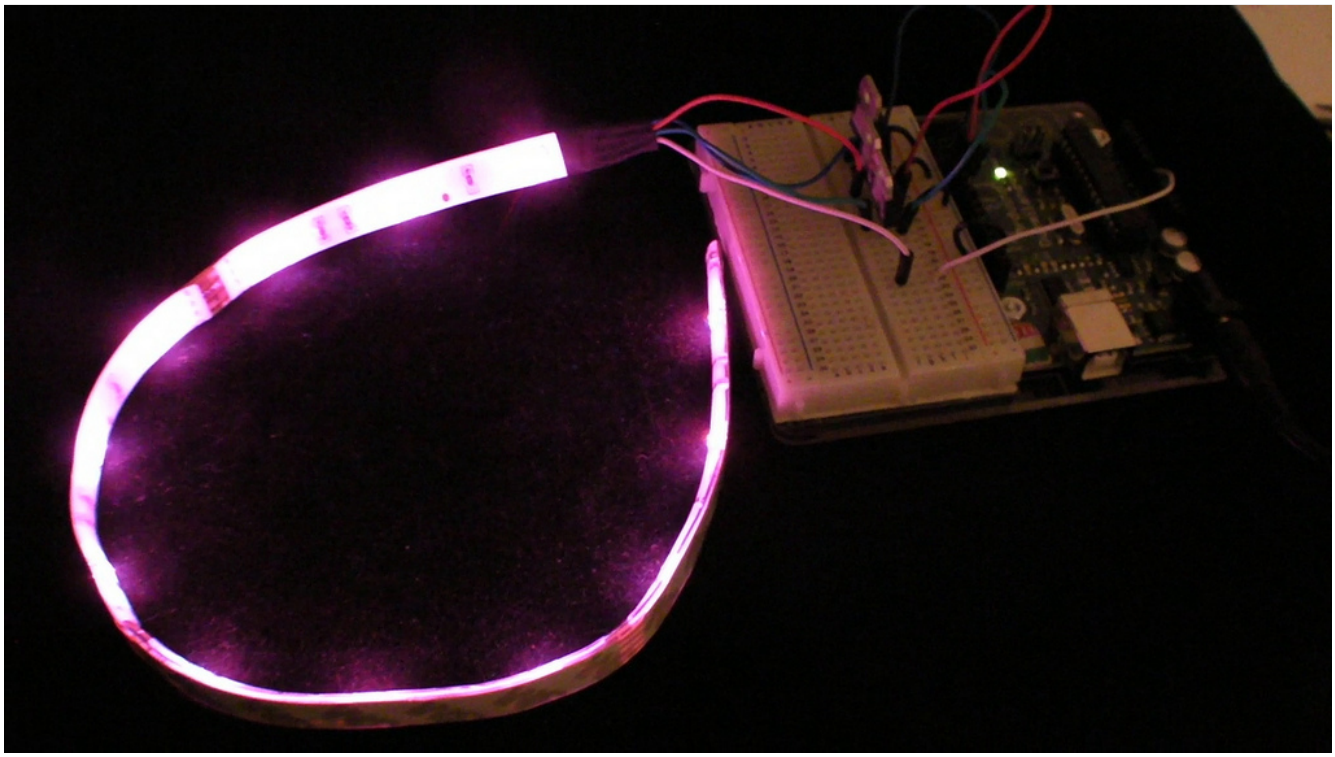
RGB LED Strips

Created by lady ada

Last updated on 2017-11-26 10:21:20 PM UTC

Guide Contents

Overview



We love some good LED blinking as much as the next person but after years of LED-soldering we need something cooler to get us excited. Sure there are RGB LEDs and those are fun too but what comes after that? Well, we have the answer: **LED Strips**! These are *flexible* circuit boards with full color LEDs soldered on. They take a lot of LED-wiring-drudgery out of decorating a room, car, bicycle, costume, etc. The ones we carry are also waterproof (although not all are).

There are two basic kinds of LED strips, the "analog" kind and "digital" kind. Analog-type strips have all the LEDs connected in parallel and so it acts like one huge tri-color LED; you can set the **entire** strip to any color you want, but you can't control the individual LED's colors. They are very very easy to use and fairly inexpensive.

The Digital-type strips work in a different way. They have a chip for each LED, to use the strip you have to send digitally coded data to the chips. However, this means you can control each LED individually! Because of the extra complexity of the chip, they are more expensive.

You can buy waterproof analog-type RGB LED strips by the meter at the Adafruit shop!

> This tutorial is for the Analog RGB LED strips only!

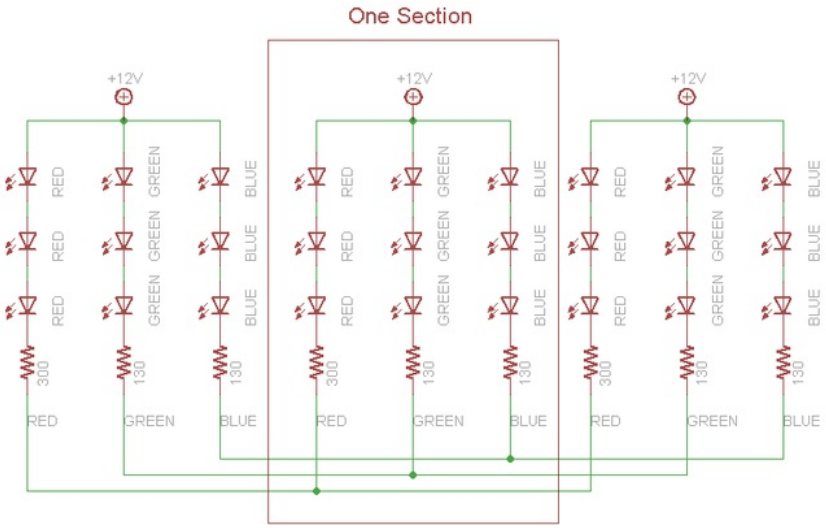
Technical specs:

- 10.5mm (0.41") wide, 3mm (0.12") thick, 100mm (3.95") long per segment
- Clear waterproof molded
- 3M adhesive strip on back
- Maximum 12V @ 60mA draw per strip segment
- 3 common-anode RGB LEDs per segment
- LED wavelengths: 630nm/530nm/475nm
- No microcontroller or chip controller ('analog' only!)
- (We're working on getting an English datasheet from the manufacturer!)


+12V
G
R
B

Schematic

Analog type RGB LED strips come on a reel, and are made of 3-LED sections that are 10 cm long. They are easy to cut at the boundary of each section, theres a little cut mark area and some copper tabs you can solder to. Each LED in a section is a '5050' tri-color type, containing a red, green and blue LED. That means that every section really has 9 total LEDs - three red, three green and three blue. The LEDs are arranged in series as shown in the following schematic:



Current Draw

Because there are three LEDs in series, you cannot drive these LEDs from a 5V supply. The LED strips say "+12V" on them to mark the anode and that's the maximum voltage we suggest. We've found that if you're ok with them being a little dimmer, even 9VDC works very well.

Each segment of 3 LEDs draws **approximately** 20 milliAmperes from a 12V supply, per string of LEDs. So for each segment, there is a maximum 20mA draw from the red LEDs, 20mA draw from the green and 20mA from the blue. If you have the LED strip on full white (all LEDs lit) that would be 60mA per segment.

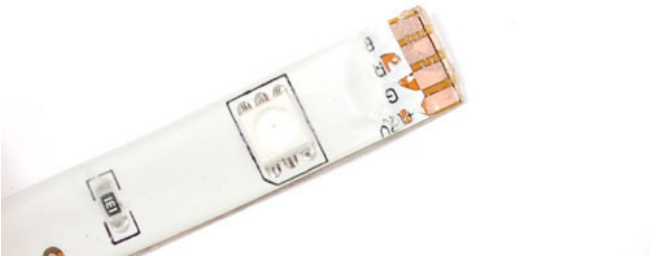
To find the total maximum current draw per meter, we would multiply **60mA x 10** (ten segments per meter for the 30/LED per meter strip) =**0.6 Amps per meter** OR **60mA x 20** (twenty segments per meter for the 60/LED per meter strip) =**1.2 Amps per meter**. Again, that's assuming you would have all the LEDs on at once and that you are powering it from 12V. If you're going to be PWM-fading between colors, maybe 1/2 of that is what you'll be drawing. Still, you do need to have a fairly decent power supply to run this strip, all those LEDs add up!

Wiring

Connecting up to the strip is fairly easy, you'll want to solder four wires to the copper tabs. We'll use white for +12V, then red, green and blue wires for the corresponding LED colors.

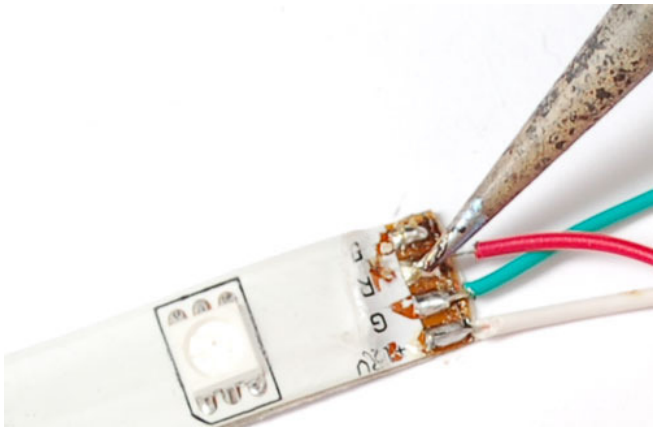
Cut away the waterproof overmolding at one end of the strip. The strips are symmetric so it doesn't matter which end you use.

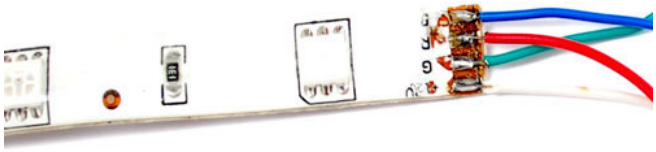
Scrape away the rubber to expose the copper pads.



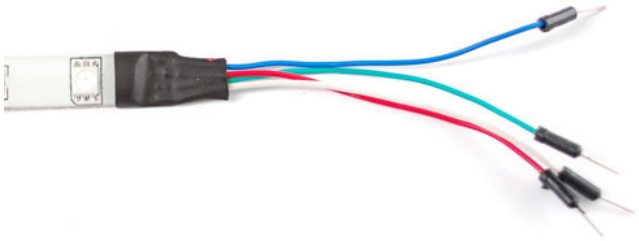
Melt some solder onto the pads to tin them and also burn away any left over rubber.

Solder the four wires on. We used stranded wire, which is more flexible and is probably a better choice than solid-core.





To protect the wires and maintain some waterproofness, you can use heatshrink.



Usage

Because these LED strips are very simple, we can easily use them with any microcontroller. We suggest using PWM dimming techniques to control the strip. Since each 'LED' pin may end up requiring an Amp or more to sink to ground, power transistors are **required**! Don't try to connect the pins directly to your everyday microcontroller, they will burn out and/or not work.

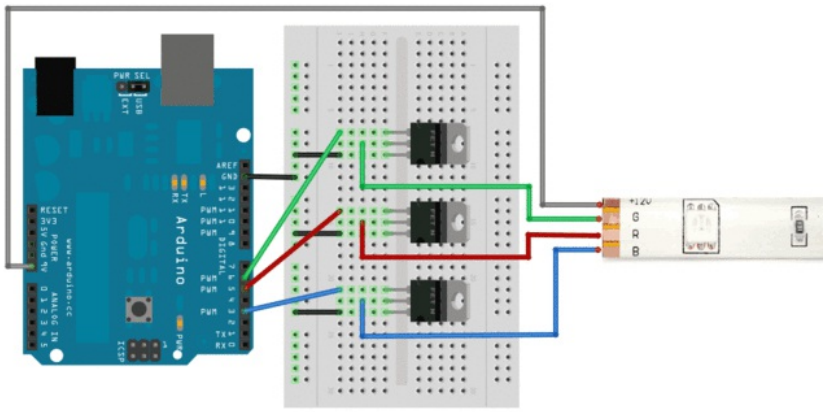
You can use any power NPN or N-Channel MOSFET, make sure the transistor is rated to be able to pass as much current as you need. For example, since we draw about 0.2Amps per channel per meter, if you have a 5 meter strip you will need to pass up to 1 Ampere per transistor. Get the beefy "TO-220" packages, not the dinky little guys. Make sure they look like this:

For basic, low-cost usage we suggest using N-channel MOSFETs such as the IRLB8721 - they are very popular and inexpensive and work with 3.3V or 5V logic. If you can't get those, TIP120 are also good but there is more voltage loss in a transistor than in a MOSFET which is why we suggest those first (less heat loss, more light!)

This diagram shows connecting up with N-Channel MOSFETs where the Gate is pin 1, the Drain is pin 2 and the Source is pin 3

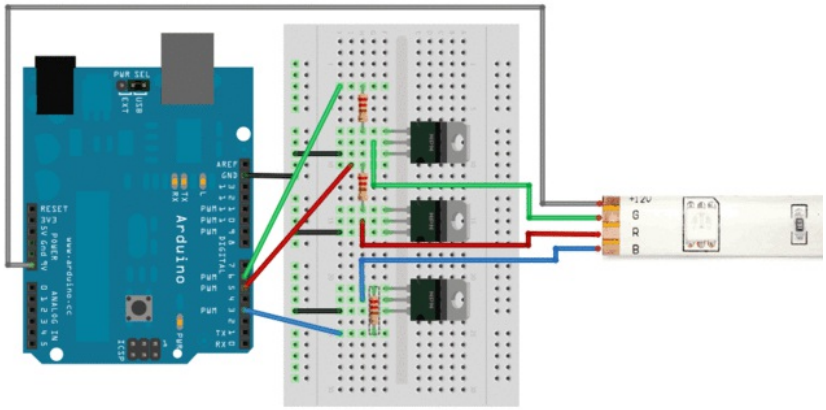
The **IRLB8721**'s can handle up to **16 Amps** of continuous current - so that's at least 750 LEDs, and if you don't have them all on bright white, 1500 LEDs. You may need to heat sink depending on the continuous/overall power draw/dissipation

For longer strips requiring more than 1 Amp, wire the power directly to the strip, then run power and ground wires back to the Arduino.



This diagram shows connecting up with power NPN transistors such as TIP120, where Base is pin 1, Collector is pin 2 and Emitter is pin 3. Its very similar except this time we have 100-220 ohm resistors between the PWM output pin and the base.

For longer strips requiring more than 1A, wire power directly to the strip, then run power and ground wires back to the Arduino.



Connect a 9-12V power supply to the Arduino so that **Vin** supplies the high voltage to the LED strip. If you want, you can also just use a separate wire that connects to a power supply that provides about +12V. Make sure to connect the ground of that supply to the ground of the Arduino/MOSFETs!

TIP120's can handle up to **5 Amps** of continuous current - so that's at least 250 LEDs, and if you don't have them all on bright white, 500 LEDs.

Arduino Code

Once you have the strip wired up, it is easy to control the color of the strip by using PWM output, for Arduino you can use **analogWrite()** on pins 3, 5, 6, 9, 10 or 11 (for classic Arduinos using the Atmega328 or 168). An **analogWrite(pin, 0)** will turn that LED off, **analogWrite(pin, 127)** will turn it on half-way and **analogWrite(pin, 255)** will turn it on full blast. Here is some example code that performs a simple color-swirl.

If you want to use other pins, check out this page on analogWrite() to know which models support analogWrite() on which pins

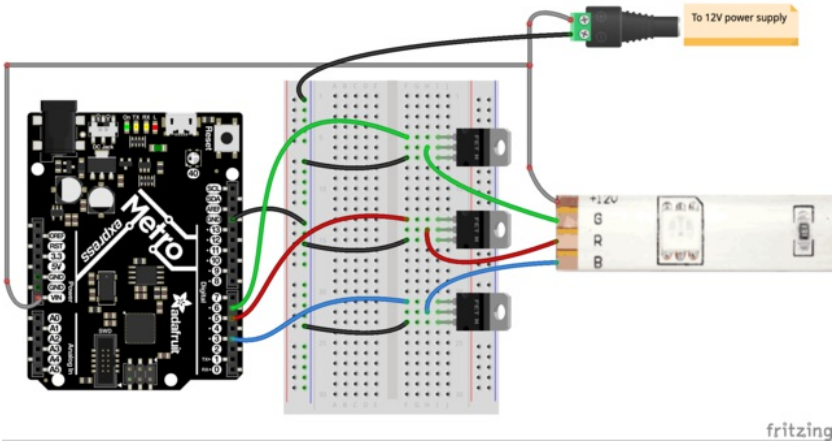
```
// color swirl! connect an RGB LED to the PWM pins as indicated
// in the #defines
// public domain, enjoy!

#define REDPIN 5
#define GREENPIN 6
#define BLUEPIN 3

#define FADESPEED 5     // make this higher to slow down

void setup() {
  pinMode(REDPIN, OUTPUT);
  pinMode(GREENPIN, OUTPUT);
  pinMode(BLUEPIN, OUTPUT);
}


void loop() {
  int r, g, b;

  // fade from blue to violet
  for (r = 0; r < 256; r++) {
    analogWrite(REDPIN, r);
    delay(FADESPEED);
  }
  // fade from violet to red
  for (b = 255; b > 0; b--) {
    analogWrite(BLUEPIN, b);
    delay(FADESPEED);
  }
  // fade from red to yellow
  for (g = 0; g < 256; g++) {
    analogWrite(GREENPIN, g);
    delay(FADESPEED);
  }
  // fade from yellow to green
  for (r = 255; r > 0; r--) {
    analogWrite(REDPIN, r);
    delay(FADESPEED);
  }
  // fade from green to teal
  for (b = 0; b < 256; b++) {
    analogWrite(BLUEPIN, b);
    delay(FADESPEED);
  }
  // fade from teal to blue
  for (g = 255; g > 0; g--) {
    analogWrite(GREENPIN, g);
    delay(FADESPEED);
  }
}
```

CircuitPython Code

You can use analog RGB LED strips with CircuitPython's built-in analog/PWM output modules. Just like with dimming a LED's brightness using CircuitPython you can use PWM outputs to control the brightness of the strip's red, green, and blue LEDs. This allows you to set the color of the strip to anything you can imagine!

Just like with wiring to an Arduino you need to use transistors to buffer and control the higher current sent to the LED strips (your board can't supply all that current itself!). Be sure to follow the usage page to wire the LED strip to your board with power transistors as shown. This guide assumes the same wiring with board pin 5 connected to the strip's red LEDs, board pin 6 connected to green, and board pin 3 connected to blue. As mentioned on the usage page too be sure to use a powerful 12V external power supply with enough current to drive all the LEDs.

Here's an example of a Metro M0 Express wired to MOSFETs that drive a strip of LEDs:



Fritzing Source

https://adafru.it/A3B

Note: Be sure to use PWM output pins from your board! Not all digital inputs/outputs support PWM output. Check your board's guide and pinout to confirm. Typically a PWM-capable output will have a dot or wavy signal line next to it on the board's silkscreen.

Also be very careful to ensure your board can be powered by 12 volts! If you're driving your board from the same 12 volt supply as your LED strip you need to make sure you connect the 12 volt power to the appropriate pin on your board so it is properly regulated down to the necessary logic levels for your board. In particular be careful about using a Feather and 12 volt power, there's no Vin line like on Metro/Arduino boards so it's trickier to power them with higher voltage levels like 12 volts and you will **damage the boards**. **Don't power a Feather, Trinket or Gemma from 12V or it can damage the board, use a Metro board instead!**

Now connect to the board's serial REPL so you are at the CircuitPython >>> prompt.

Import the necessary **pulseio** and **board** modules by running:

```
import board
import pulseio
```

Then create a PWM output for each LED connection (red, green, blue):

```
red = pulseio.PWMOut(board.D5)
green = pulseio.PWMOut(board.D6)
blue = pulseio.PWMOut(board.D3)
```

Like the analog I/O guide mentions you can change the duty cycle of each PWM output to control the brightness of the LEDs. A larger duty cycle means the LEDs are turned on for longer and appear brighter. Remember the duty cycle is a value that goes from 0 (0% / not turned on) to 65535 (100% / turned on all the time).

To set the strip to a purple color with 100% red, 0% green, and 50% blue you could run:

```
red.duty_cycle = 65535   # 100% red
green.duty_cycle = 0     # 0% green
blue.duty_cycle = 32767  # 50% blue
```

Like the analog I/O guide mentions though it might be easier to make a function that converts a percent duty cycle value to the numeric value and use that to simplify your code. Here's how to define and use the function to set the same purple color:

```
def duty_cycle(percent):
    return int(percent / 100.0 * 65535.0)

red.duty_cycle = duty_cycle(100)  # 100% red
green.duty_cycle = duty_cycle(0)  # 0% green
blue.duty_cycle = duty_cycle(50)  # 50% blue
```

Here's a complete example of using CircuitPython to control the LED strip and fade it between different colors just like with the similar Arduino code from the previous page. Save this as a **main.py** on your board's root filesystem and it will fade the colors of the strip when the board starts:

```
import board
import pulseio
import time

RED_PIN = board.D5    # Red LED pin
GREEN_PIN = board.D6  # Green LED pin
BLUE_PIN = board.D3   # Blue LED pin

FADE_SLEEP = 10  # Number of milliseconds to delay between changes.
                 # Increase to slow down, decrease to speed up.

# Define PWM outputs:
red = pulseio.PWMOut(RED_PIN)
green = pulseio.PWMOut(GREEN_PIN)
blue = pulseio.PWMOut(BLUE_PIN)

# Function to simplify setting duty cycle to percent value.
def duty_cycle(percent):
    return int(percent / 100.0 * 65535.0)

# Fade from nothing up to full red.
for i in range(100):
    red.duty_cycle = duty_cycle(i)
    time.sleep(FADE_SLEEP)

# Now fade from violet (red + blue) down to red.
for i in range(100, -1, -1):
    blue.duty_cycle = duty_cycle(i)
    time.sleep(FADE_SLEEP)

# Fade from red to yellow (red + green).
for i in range(100):
    green.duty_cycle = duty_cycle(i)
    time.sleep(FADE_SLEEP)

# Fade from yellow to green.
for i in range(100, -1, -1):
    red.duty_cycle = duty_cycle(i)
    time.sleep(FADE_SLEEP)

# Fade from green to teal (blue + green).
for i in range(100):
    blue.duty_cycle = duty_cycle(i)
    time.sleep(FADE_SLEEP)

# Fade from teal to blue.
for i in range(100, -1, -1):
    green.duty_cycle = duty_cycle(i)
    time.sleep(FADE_SLEEP)
```