adafruit learning system

36mm LED Pixels

Created by Phillip Burgess

Last updated on 2018-08-22 03:31:00 PM UTC

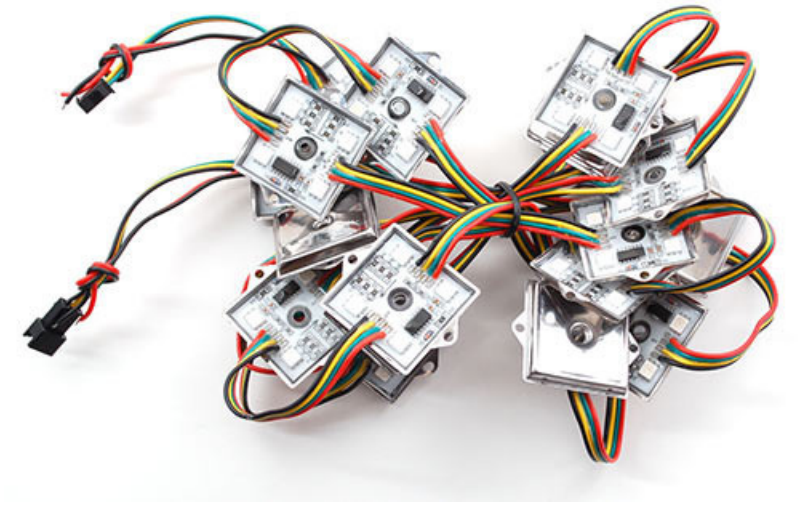
Guide Contents

Overview

RGB Pixels are digitally-controllable lights you can set to any color, or animate. Each pixel contains four RGB LEDs and a controller chip in a sturdy metal housing. The pixel is then 'flooded' with epoxy to make it waterproof. These are fairly large pixels but they have a lot of nice mounting options, such as two metal flanges on the side and a 0.15"/4mm diameter hole in the middle so you can screw them directly onto a surface. They're typically used to make outdoor signs. Compared to our other LED dots, these are much bigger and much brighter, good for larger scale installations.

At **12 volts**, they draw a maximum of **120 milliamps** per pixel: 40 mA each for red, green and blue.

The LED pixels are spaced along a strand of ribbon cable, with about **3 inches (75mm)** between pixels. If additional distance is needed you can cut the ribbon cable and solder 4 wires to extend the gap to the desired length.



Each pixel contains a small microchip. The WS2801 LED driver chip is custom designed for this purpose. We provide an Arduino library for communicating with the pixels (explained in subsequent pages), but if you want to write your own code for other microcontrollers, they're very easy to communicate with using an SPI-like protocol. For each pixel, one "shifts out" 24 bits of color information — the first data out corresponds to the pixel closest to the microcontroller. To write colors to 10 LEDs, you would issue 240 bytes (10 * 24). Following the data, a 500 microsecond pause will then "latch" the data and display the new LED colors.

Project Ideas

These pixels could be used for stuff like...

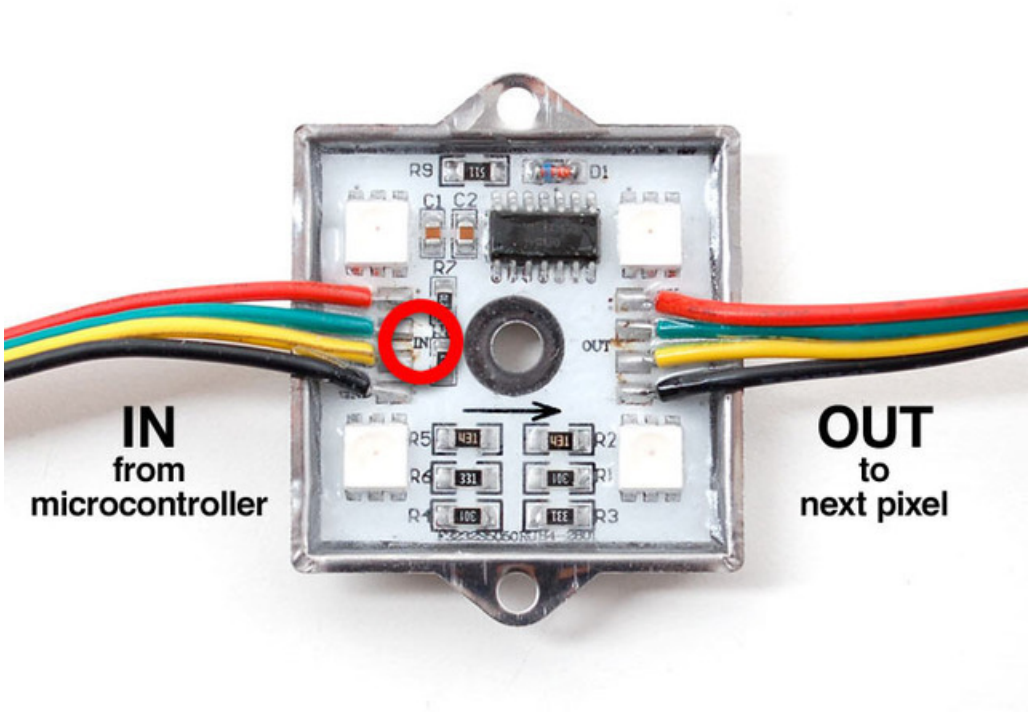
An LED coffee table (this one was “hand made” some time ago...now you could simply use a long strand of our LED pixels and skip the complicated wiring and driver parts):

Signs and displays:

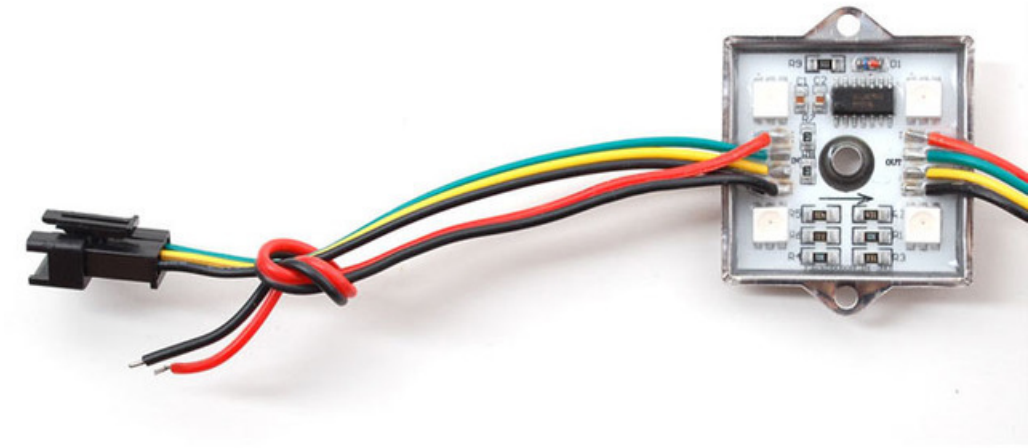
Wiring

The “magic” of these pixels is that they're digitally controlled...even though there are only two control lines, you can have as many pixels as you’d like in a single long strand, yet each remains independently controllable.

Though it looks like the 4-conductor ribbon cable is continuous, *it isn't!* The pixels have a distinct “in” and “out” side. Data from the microcontroller arrives on the input side, where it’s received by the driver chip. The output side then connects to the input of the next pixel, all the way down the line.
When connecting these pixels to a microcontroller, make sure you're connecting to the strand's **input** pins! On these large pixels, it's easy to spot: examine the circuit board, looking for the "IN" label and an arrow indicating the direction of data flow. If connecting multiple strands together, make sure the output of one strand goes to the input of the next.

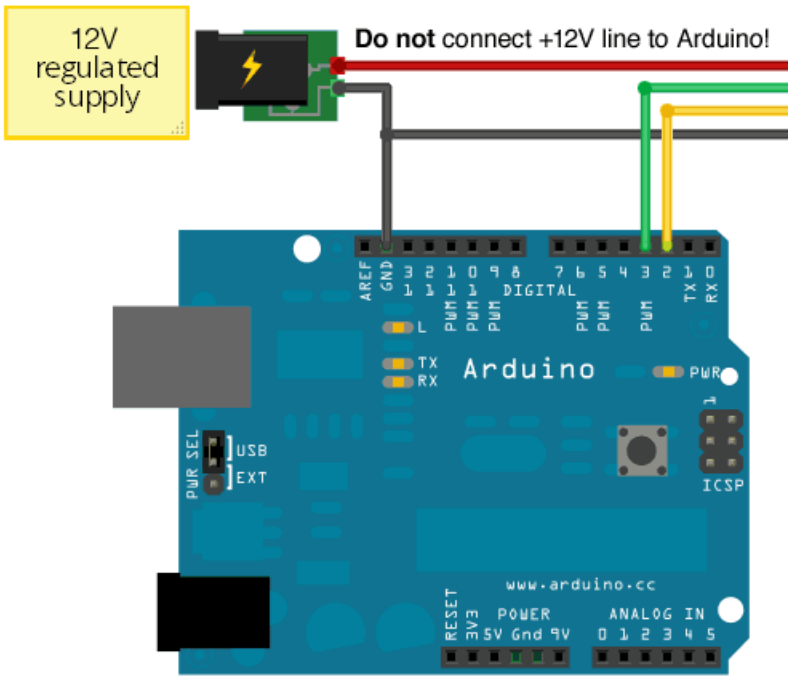


The pixel strands include plugs for joining multiple strands, plus two wires for connecting power:



Wiring is pretty easy since there are only 4 wires. The only important thing is that you should not try to power the LED strand from the Arduino's 5V line — these LEDs require a dedicated 12V source separate from the microcontroller.

Use this diagram with the red wire going to +12V from the power supply, green (serial clock) to Arduino digital pin 3, yellow (serial data) to digital pin 2, and black to both the ground connection on the power supply and any available GND pin on the Arduino.



Our Arduino library can use any two pins, but the examples are written to use pins 2 and 3 as above.

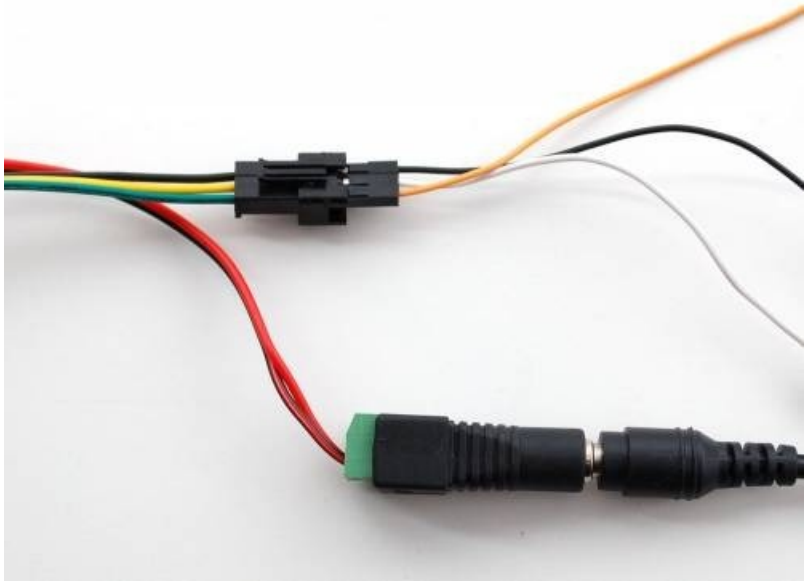
Powering

When running a lot of LEDs, it's important to keep track of power usage. Individual LEDs don't get very hot or use tons of power, but they add up fast!

Each single 36mm RGB LED pixel can draw up to 120mA from a 12V supply. That means a strand of 20 can use up to 2.4 Amps. That's a peak rate, which assumes that all the LEDs are on at full brightness. If most of the LEDs are kept dim or off (as when animating patterns), the average power usage can be 1/3 this or less.
We suggest a nice switching supply for driving LED pixels, such as our12 Volt 5 Amp supply (http://adafru.it/352) . For larger projects, a slightly modified ATX computer power supply (https://adafru.it/aHR) can provide 10 Amps to power upwards of 80 pixels!

Since the 36mm pixels are powered by 12V but use 3-5V signaling, make sure you don't accidentally connect the 12V to your microcontroller — this will destroy it! For that reason, the power lines are separated out. Use something like a 2.1mm DC terminal block adapter (http://adafru.it/368) so that you can directly plug in our 12V adapter.

The wires on the 3-JST SM connector are then connected to your microcontroller: black is ground, yellow is data and green is clock.



Code

Installation

To download the Arduino library, visit the repository on GitHub (https://adafru.it/aHV) . Click the DOWNLOAD ZIP button near the upper left, extract the archive and then rename the uncompressed folder to Adafruit_WS2801. Confirm that this folder contains the files Adafruit_WS2801.cpp and Adafruit_WS2801.h and the examples folder.

Place the Adafruit_WS2801 folder inside your Arduino libraries folder. You may need to create this folder if it does not yet exist. In Windows, this would be (home folder)\My Documents\Arduino\Libraries and for Mac or Linux is (home folder)/Documents/Arduino/Libraries. We also have a tutorial on library installation (https://adafru.it/aYG).

After installing the Adafruit_WS2801 library, restart the Arduino IDE. You should now be able to access the sample code by navigating through menus in this order: File→Sketchbook→Libraries→Adafruit_WS2801→strandtest

Using the Library

Let's look through the strandtest example code. To use the library in an Arduino sketch, you'll first need to globally declare a WS2801 object to talk to the strip. It is invoked with three variables: the number of pixels and the data and clock pins:

```
int dataPin = 2;
int clockPin = 3;

// Set the first variable to the NUMBER of pixels. 25 = 25 pixels in a row
Adafruit_WS2801 strip = Adafruit_WS2801(25, dataPin, clockPin);
```

The example code was written for multiple types of pixels we carry, some of which come in different strand lengths. Since the 36mm strands contain 20 (rather than 25) pixels, feel free to edit the declaration for 20 pixels:

```
Adafruit_WS2801 strip = Adafruit_WS2801(20, dataPin, clockPin);
```

Next, we initialize the strip in the setup() procedure:

```
void setup() {

  strip.begin();

  // Update LED contents, to start they are all 'off'
  strip.show();
}
```

begin() initializes the library, while show() refreshes the displayed colors of the LEDs. You'll need to call show() after changing any pixel colors to see this reflected in the LEDs. This way you can change the entire strip at one time (it takes the same amount of time to change one pixel as it does for an entire strip).

Let's look inside an example function, colorWipe(). This creates a 'chase' sequence that fills the strip with a color. It is basically a loop that increments through every pixel (which you can query with the numPixels() function) and sets the color of each (incremented with i) to the value passed (c — colors are expressed as a 32-bit variable type, though only the bottom 24 bits are used). The strip output is then updated with show(). Finally there is some delay (otherwise this would happen instantly).

Below that is a helper function that converts a color from separate 8-bit red, green and blue values into a combined 24-bit value (suitable for passing to colorWipe()). The brightness range is from 0 (off) to 255 (max brightness).

```
// fill the dots one after the other with said color
// good for testing purposes
void colorWipe(uint32_t c, uint8_t wait) {
  int i;

  for (i=0; i < strip.numPixels(); i++) {
      strip.setPixelColor(i, c);
      strip.show();
      delay(wait);
  }
}

/* Helper functions */
// Create a 24 bit color value from R,G,B
uint32_t Color(byte r, byte g, byte b)
{
  uint32_t c;
  c = r;
  c <<= 8;
  c |= g;
  c <<= 8;
  c |= b;
  return c;
}
```

For example, in the loop() function we call colorWipe(Color(255, 0, 0), 50) which will fill the strand with full-brightness red light, pausing about 50 milliseconds between pixels.

```
  colorWipe(Color(255, 0, 0), 50);  // red fill
  colorWipe(Color(0, 255, 0), 50);  // green fill
  colorWipe(Color(0, 0, 255), 50);  // blue fill
```

Troubleshooting

Many support issues arise from eager users getting ahead of themselves, changing the code and wiring before confirming that all the pieces work in the standard configuration. We recommend always starting out with the examples as shown. Use the pinouts and wiring exactly as in the tutorial, and run the stock, unmodified "strandtest" example sketch. Only then should you start switching things around.

Here are the most common issues and solutions...

The pixels are wired and powered exactly as in the tutorial, the sketch compiles and uploads successfully, but nothing happens.

- Double-check all wiring. Are the clock and data wires swapped? Is ground connected to the Arduino?
- Confirm the Arduino is connected to the INPUT end of the strand.
- Check power supply polarity and voltage. Are + and - swapped? If you have a multimeter, confirm 12V DC output (±10%) from the power supply.
- Are the power wires at the opposite end of the strand insulated or trimmed? They should not be left exposed where they might make contact with metal, or each other.
- Is the correct board type selected in the Arduino Tools→Board menu?

A few LEDs randomly turn on when power is applied, but then nothing happens.

The power supply is probably OK. Check for any of the following:

- Double-check all wiring. Are the clock and data wires swapped? Is ground connected to the Arduino?
- Confirm the Arduino is connected to the INPUT end of the strand.
- Is the correct board type selected in the Arduino Tools→Board menu?
- Did the strandtest code successfully compile and upload?

Only the first few LEDs respond. The rest remain off or flicker randomly.

- Confirm that the number of LEDs in the Adafruit_WS2801() constructor match the number of LEDs in the strand (both will be 20 if using the strandtest example and a single strand of LEDs).

The LEDs flicker randomly, not the way they're programmed to.

Are the clock and data wires swapped? Is ground connected to the Arduino?

"White" LEDs are showing pink instead.

- This can happen when trying to power too long of a strand from one end. Voltage will drop along the length of the strand and the furthest pixels will "brown out." Connect power to *every* 20 pixel strand.

Sketch will not compile: error message is "Adafruit_WS2801 does not name a type"

- Confirm the library is unzipped prior to installation.
- Confirm the library is properly named and located. The folder should be called Adafruit_WS2801, and placed inside your personal Documents/Arduino/Libraries folder — *not* inside the Arduino application folder!
- After installation, the Arduino IDE needs to be restarted for new libraries to be used.