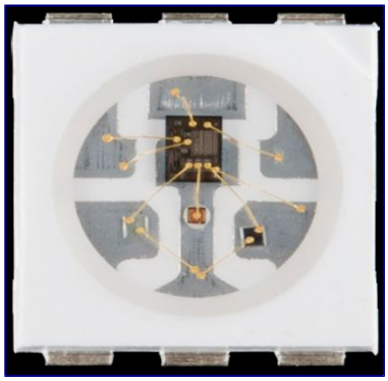
This 1-meter long strip contains 60 RGB LEDs that can be individually addressed using a one-wire interface, allowing you full control over the color of each RGB LED. The flexible, waterproof strip runs on 5 V and can be chained with additional WS2812B strips to form longer runs or cut apart between each LED for shorter sections.

WS2812

The WS2812 is much more than meets the eye. It may look like a common 5050-sized (5x5mm) LED, but there's actually an integrated circuit embedded inside there too. If you look really hard, you can see the tiny black chip hidden in there, along with minuscule gold wires connecting the chip to the LED.

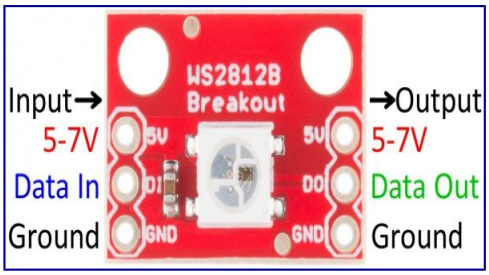


Pretty nifty view at the guts of the WS2812.

The LED itself is like any RGB (Red/Green/Blue) LED. The brightness of each color can be adjusted using pulse-width modulation to one of 256 different levels. That means there are 16,777,216 (256^3) possible combinations of colors. You can produce any color from white to black (off), or salmon to sienna.

Breakout Board Pinout

The Breakout board mounts that multi-taltented LED onto a PCB, and breaks out the few pins required to control the LED.



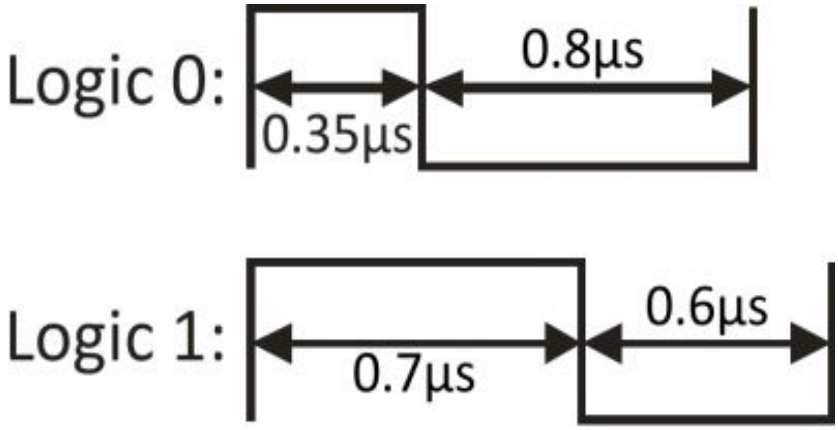
Four unique pins are broken out:

- **5V** – This should be a **regulated** supply voltage between 5V and about 7V. More than that could harm the LED, less than 5V will either reduce brightness, or it just won't turn on.
- **GND** – The common, ground, 0V reference supply voltage.
- **DI** – Data from a microcontroller (or another WS2812 pixel) comes into this pin.
- **DO** – Data is shifted out of this pin, to be connected to the input of another pixel or left floating if it is the last link in the chain.

Data Transmission Interface

Note: this stuff is ugly, and not critical to understand if you just want to use the breakout board. (That's what libraries are for, right?!) It's interesting to talk about, because the interface is so unique.

The communication interface between a microcontroller and the WS2812 is weird. It's one wire, but it's not like a standard, UART serial interface. This interface is very time-specific. Both a logic 0 and a logic 1 require a square pulse, and it's the length of the pulse that defines which it is.



Timing diagram for a single bit of value 0 or 1.

The data is sent in a sequence containing 24 of those bits – 8 bits for each color – followed by a low "reset" pulse of at least 50µs.

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

A sequence of 24 timed-bits sets the color for each channel. 8-bits per channel. Green first, then red, then blue.

The larger the value of a specific color is, the brighter it will be. If every color is set to 0, the LED will be off. If every color is set to max – 255 – the LED will be as bright and white as can be.

This is all to say that the interface is **very time-specific**. To run the LEDs you'll need a real-time processor, like an Arduino; microprocessors like those on the Raspberry Pi or pcDuino can't give you a reliably-timed pulse. Even if one bit is less than a microsecond off, that could mean the difference between purple and maroon.