Adafruit NeoPXL8 FeatherWing and Library

Created by lady ada

https://learn.adafruit.com/adafruit-neopxl8-featherwing-and-library

Last updated on 2023-01-05 12:54:36 PM EST

Table of Contents

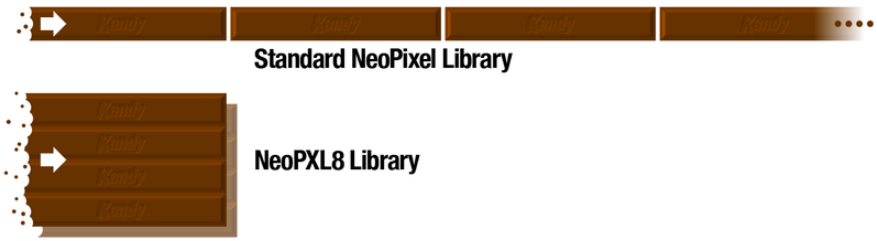
Overview

NeoPixels are magical things. It couldn't be simpler...a single data wire from the microcontroller, linking pixel to pixel for as long as you need. When NeoPixel projects get really large though...hundreds of pixels or more...this simplicity starts to become a bottleneck...

- The standard NeoPixel data rate is a fixed 800 KHz, or 30 microseconds per 24-bit pixel. As projects scale into the hundreds or thousands of pixels, the time spent issuing all that data reaches progressively larger fractions of a second; animation becomes less smooth.
- While this data is being transmitted, all other processing on the microcontroller s tops, including interrupts which keep track of time. This is why the millis() and micros() functions gradually drift in NeoPixel projects.

NeoPXL8 (pronounced "NeoPixelate") is a hardware-and-software combo that works around these limitations to bring buttery smooth animation to large-scale NeoPixel projects.

NeoPXL8 splits the problem 8 ways: rather than one long strand of, say, 1,000 pixels*, eight strands of 125 pixels can operate concurrently in perfect sync. It's like shoving two whole Kit Kats® in your mouth "the illegal way" instead of nibbling one bar at a time. Data transmission times are greatly reduced and animation can remain smooth.



* Hypothetical situation, not an imposed limit. Depends on available RAM, but most "M0" (SAMD21) boards might handle upwards of 2,500 pixels, with "M4" (SAMD51) and RP2040 potentially up to 15,000...but in reality, you'll want some fraction of that, so your code has time to compute those pixels.

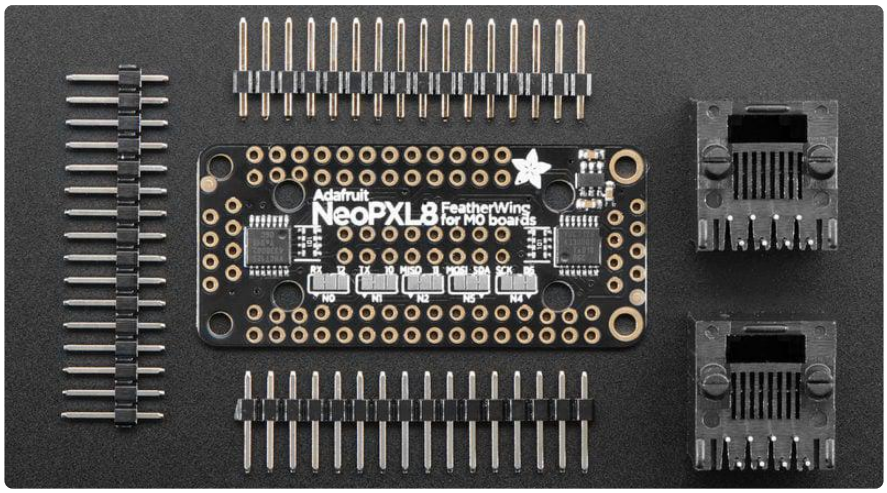
Additionally, NeoPXL8 uses direct memory access (DMA) to allow the CPU to continue with other tasks while these data transfers take place in the background. Your code could start processing the next frame of animation, or load data from an SD card. All interrupts and timekeeping functions operate normally, no drift.

The NeoPXL8 code relies on features unique to the SAMD21, SAMD51, RP2040 and ESP32-S3 microcontrollers — it should work on any Adafruit "M0" or "M4" board, the Arduino Zero, most RP2040 boards like the Raspberry Pi Pico, and most ESP32-S3 boards (but not S2, C3 or original ESP32). It will not work on other architectures.

The NeoPXL8 FeatherWing adapters provide 8 NeoPixel outputs with 5-Volt logic level shifting, and it stacks directly atop any of our M0 or M4 Feather boards — Basic Proto, Adalogger and so forth. (Note there are different versions for Feather M0 vs M4, they are not interchangeable!) With a minor change, the M4 version can work with the Feather RP2040. Either one works with Feather ESP32-S3.

The NeoPXL8 Friend breakout board provides similar functionality in a non-FeatherWing format, making it handy for use with boards like the Metro Express or ItsyBitsy M0 Express (and their respective M4 variants) or the Raspberry Pi Pico.

NeoPXL8 FeatherWings



For NeoPixel projects starting with one of our Feather M0 or M4 board variants, a Neo PXL8 FeatherWing can simplify and make sense of the wiring...the boards are designed to be stacked. Some soldering and “maker skills” are required.

There are two distinct versions of the NeoPXL8 FeatherWing: one for M0 boards, the other for M4. The two are not interchangeable.

You can use any Feather M0 or M4...however, some NeoPXL8 outputs get configured to take over I2C or SPI pins. In particular, if an SPI pin is used for NeoPXL8, the Adalogger, WiFi, Bluefruit will not be able to use the built in SD/ WiFi/BTLE chips! If I2C is used, most FeatherWings with I2C sensors/devices won't work!

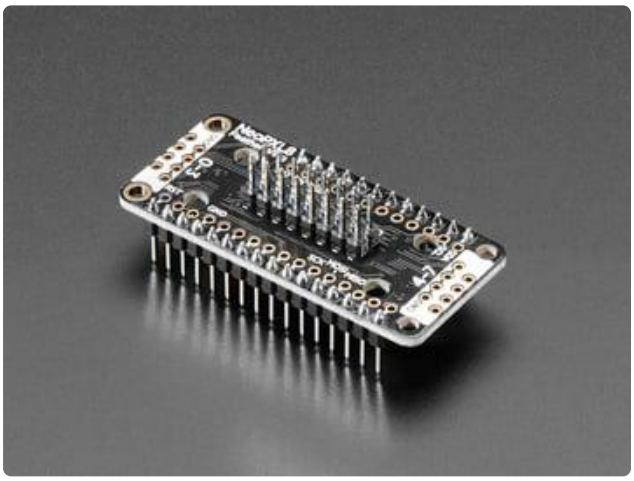
The M0 and M4 versions of the NeoPXL8 FeatherWing are NOT interchangeable — get the correct type to match your Feather board!

The M4 version of the FeatherWing can also be used with the Adafruit Feather RP2040 with just a small modification. Finish reading this page for general connection tips, then visit the “RP2040 Use” page for the fix.

Either version of the FeatherWing can also work with the Adafruit Feather ESP32-S3 without modification. But only the S3; the ESP32-S2 and original ESP32 are not supported.

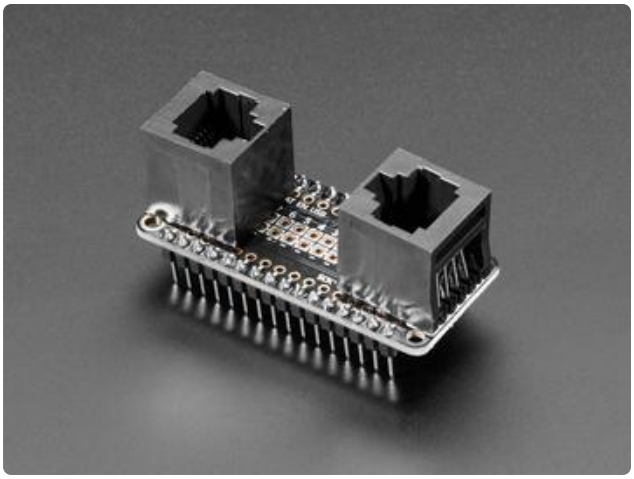
Connections: Two Types

The NeoPXL8 FeatherWing can be assembled one of two ways depending on your preferences and needs. You must choose beforehand what kind of connectors your project needs, as there's not enough room for both at once...



Populating the 8x2 row header at the center of the board (16 pins total) provides a "Fadecandy-style" connection.

Fadecandy () is a USB NeoPixel controller popular in large-scale LED installations. The 8x2 connector is fairly compact and low-profile. Assembled this way, the NeoPXL8 Feather-and-Wing combo could, with suitable Arduino code, function as a swap-in replacement in an existing Fadecandy project, or could make use of NeoPixels already wired for such.



Populating the two RJ45 connectors at the board ends provides an "OctoWS2011-style" connection.

OctoWS2811 () is a similar hardware-and-software combo for large NeoPixel setups using the PJRC Teensy 3.2 () microcontroller. These connections are bulkier but latch into place and ensure a specific polarity. Assembled this way, the NeoPXL8 Feather-and-Wing combo could, with suitable Arduino code, function as a swap-in replacement in an existing OctoWS2811 project, or could make use of NeoPixels already wired for such.

In either case, the NeoPixel headers mount on the FLAT SIDE of the NeoPXL8 FeatherWing — the side with NO COMPONENTS — and are soldered on the compone nt side. The Feather-stacking pins are done the OPPOSITE way — install from the component side, solder on the flat side. See the photos above for reference.

Additionally, you still need to build a wiring harness between these connectors and your NeoPixel LEDs. The above is just a starting point.

Adopting these two wiring schemes mean that any existing tutorials for wiring up Fadecandy or OctoWS2811 projects are applicable to NeoPXL8 as well — it's not starting over with a third incompatible standard.

This tutorial shows some Fadecandy-style wiring harnesses being made (), using a ribbon cable and 8x2 IDC header, plus lots of soldering and heat-shrink. A multimeter with continuity beep is helpful in keeping track of data wires and grounds!

The OctoWS2811 product page on the PJRC web site () shows RJ45 wiring harnesses being made by cutting open Ethernet cables.

You will also need to safely distribute 5 Volt power to all of your NeoPixels. This is not done through the NeoPXL8 board — it needs to be part of your wiring harness. This tutorial explains () some of the issues in powering large-scale NeoPixel installations.

Pin Selection

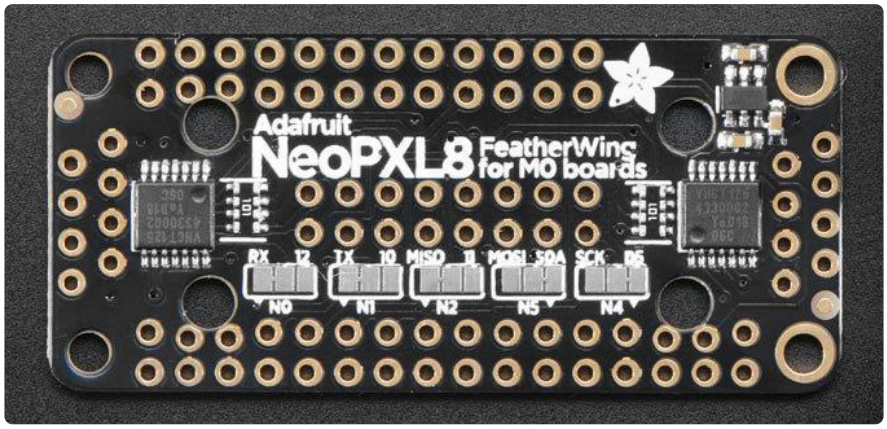
Because we're using hardware tricks, NeoPXL8 output works only on specific pins. Some are set in stone, others give some control in that you can select an alternate pin. This may be helpful if using a Feather or Wing with its own peripheral pin constraints (wireless, perhaps)...sometimes you can re-route some signals and keep full functionality.

Depending on what additional hardware you're interfacing, it's possible that neither selection will work...one peripheral or another absolutely requires that pin. In such cases, you can use NeoPXL8 with fewer than 8 outputs. We'll elaborate further on the "Library" page.

Keep track of your selections. Write it down somewhere. You'll need this information when writing Arduino sketches using the NeoPXL8 library.

The pin selection is a little different between the M0 and M4 FeatherWings...

For M0 NeoPXL8 FeatherWing only...



Let's refer to NeoPXL8's eight outputs as "0" through "7," sequentially. Outputs 3, 6 and 7 are fixed to specific Feather pins (13, A4, A3) and cannot be changed, period. Outputs 0, 1, 2, 4 and 5 offer a "this" or "that" choice.

On the component side of the FeatherWing you'll see several solder pad groups, labeled "N0", "N1", "N2", "N5" and "N4". Pay careful attention to those numbers...they are neither sequential nor contiguous (partly because three pins are fixed, partly because it was more practical to route the board this way).

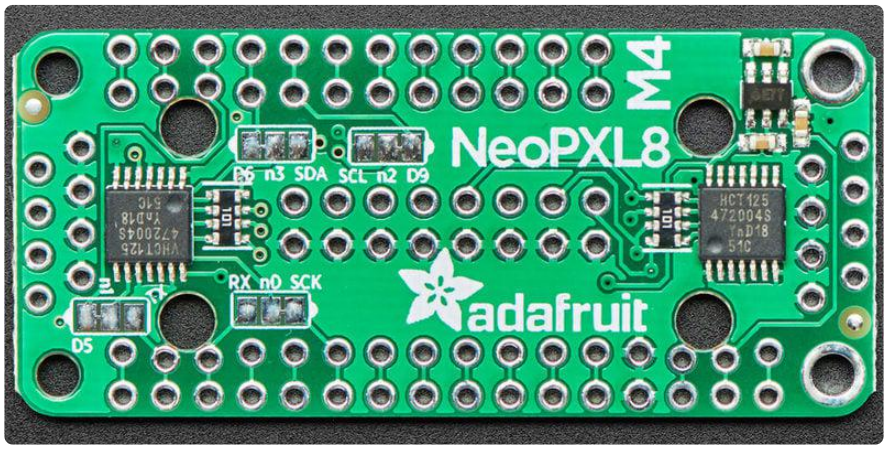
Each of these five pins has a default assignment. To change a pin to an alternate setting, use a hobby knife or file to cut the trace between the center and default pads, then apply a solder bridge between the center and alternate pad.

Output Number	Default Pin	Alternate Pin
N0	RX	12
N1	TX	10
N2	MISO	11
N5	SDA	MOSI

N4	D5 (digital pin 5, not A5)	SCK

For M4 NeoPXL8 FeatherWing only...

These rules apply only to the Feather M4 board. For Feather RP2040, see the “RP2040 Use” page.



Let’s refer to NeoPXL8’s eight outputs as “0” through “7,” sequentially. Outputs 4, 5, 6 and 7 are fixed to specific Feather pins (13, 12, 11 and 10) and cannot be changed, period. Outputs 0, 1, 2, and 3 offer a “this” or “that” choice.

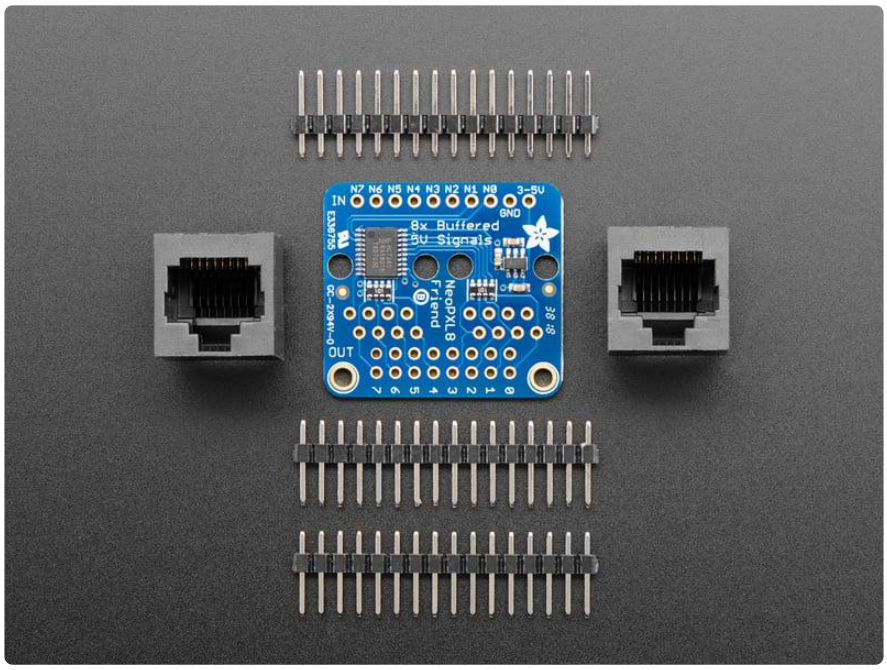
On the component side of the FeatherWing you’ll see several solder pad groups, labeled “n0”, “n1”, “n2” and “n3”.

Each of these four pins has a default assignment. To change a pin to an alternate setting, use a hobby knife or file to cut the trace between the center and default pads, then apply a solder bridge between the center and alternate pad.

Output Number	Default Pin	Alternate Pin
n0	SCK	RX
n1	D5	TX

n2	D9	SCL
n3	D6	SDA

NeoPXL8 Breakout Board



If you're working on a solderless breadboard or aren't using a Feather form-factor microcontroller, the Adafruit NeoPXL8 Friend () provides similar utility to the NeoPXL8 FeatherWing but in a non-FeatherWing package.

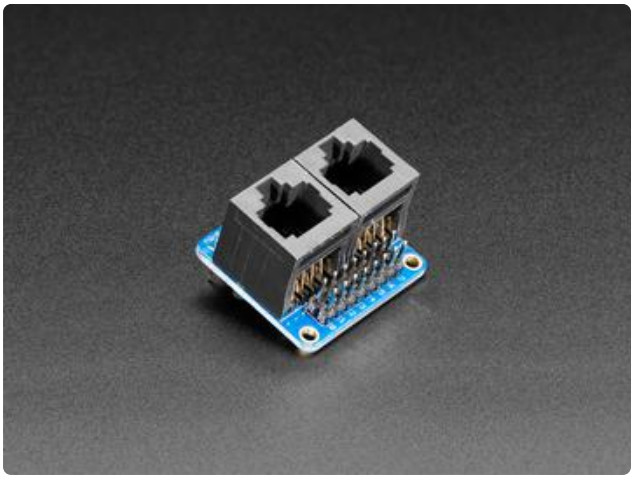
The NeoPXL8 Friend breakout board provides logic level shifting from 3V devices to as many as eight NeoPixel strips, which can be connected one of two ways:

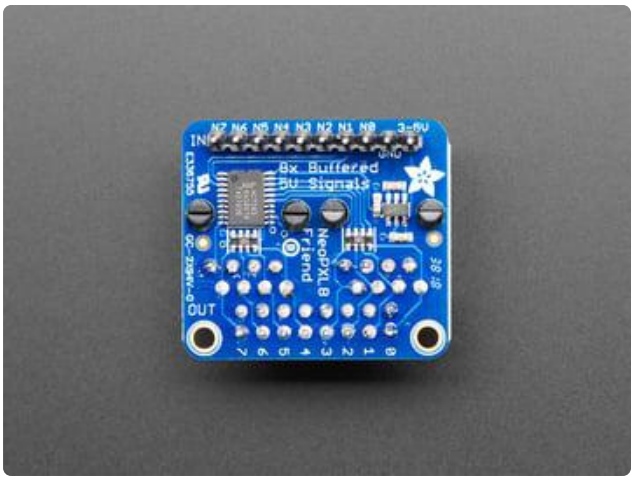
The 8x2 row header (16 pins total) provides a "Fadecandy-style" connection.

Fadecandy () is a USB NeoPixel controller popular in large-scale LED installations. The 8x2 connector is fairly compact and low-profile. Assembled this way, the NeoPXL8 Friend could, with suitable Arduino code, function as a swap-in replacement in an existing Fadecandy project, or could make use of NeoPixels already wired for such.

The two RJ45 connectors provide an "OctoWS2011-style" connection.

OctoWS2811 () is a similar hardware-and-software combo for large NeoPixel setups using the PJRC Teensy 3.2 () microcontroller. These connections are bulkier but latch into place and ensure a specific polarity. Assembled this way, the NeoPXL8 Friend could, with suitable Arduino code, function as a swap-in replacement in an existing OctoWS2811 project, or could make use of NeoPixels already wired for such.





In either case, the NeoPixel headers mount on the FLAT SIDE of the NeoPXL8 Friend — the side with NO COMPONENTS — and are soldered on the component side. The breadboarding pins are done the OPPOSITE way — install from the component side, solder on the flat side.

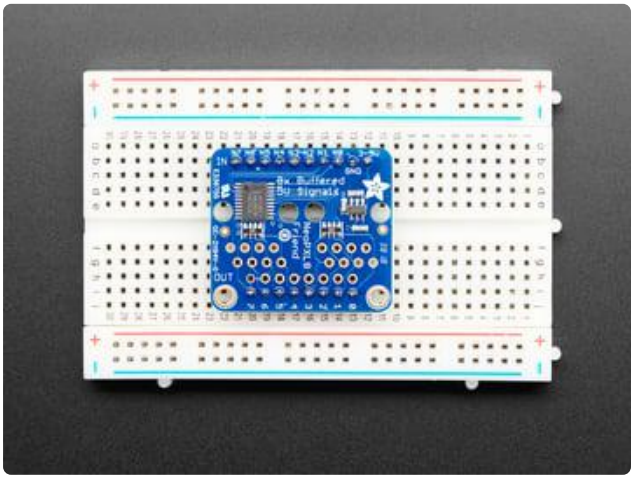
Additionally, you still need to build a wiring harness between these connectors and your NeoPixel LEDs. The above is just a starting point.

Adopting these two wiring schemes mean that any existing tutorials for wiring up Fadecandy or OctoWS2811 projects are applicable to NeoPXL8 as well — it's not starting over with a third incompatible standard.

This tutorial shows some Fadecandy-style wiring harnesses being made (), using a ribbon cable and 8x2 IDC header, plus lots of soldering and heat-shrink. A multimeter with continuity beep is helpful in keeping track of data wires and grounds!

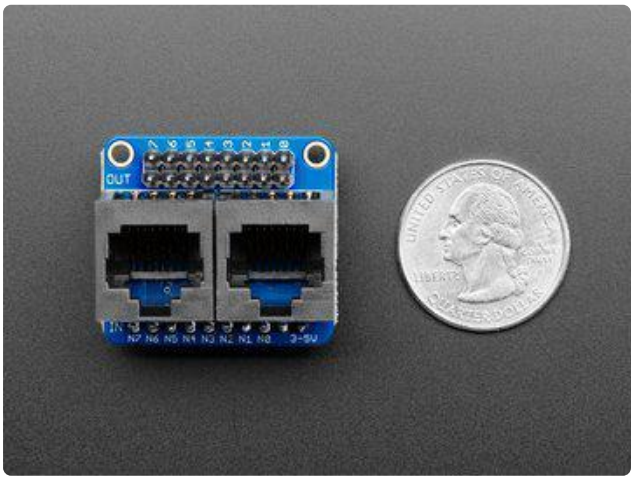
The OctoWS2811 product page on the PJRC web site () shows RJ45 wiring harnesses being made by cutting open Ethernet cables.

You will also need to safely distribute 5 Volt power to all of your NeoPixels. This is not done through the NeoPXL8 board — it needs to be part of your wiring harness. This tutorial explains () some of the issues in powering large-scale NeoPixel installations.



A third option is to install header pins on the NeoPixel outputs, so both the “ins” and “outs” are breadboardable...this may be useful for certain prototyping tasks.

There isn’t clearance for the outputs on a regular-size breadboard, but linking two breadboards side-by-side is often done with wider devices like this.



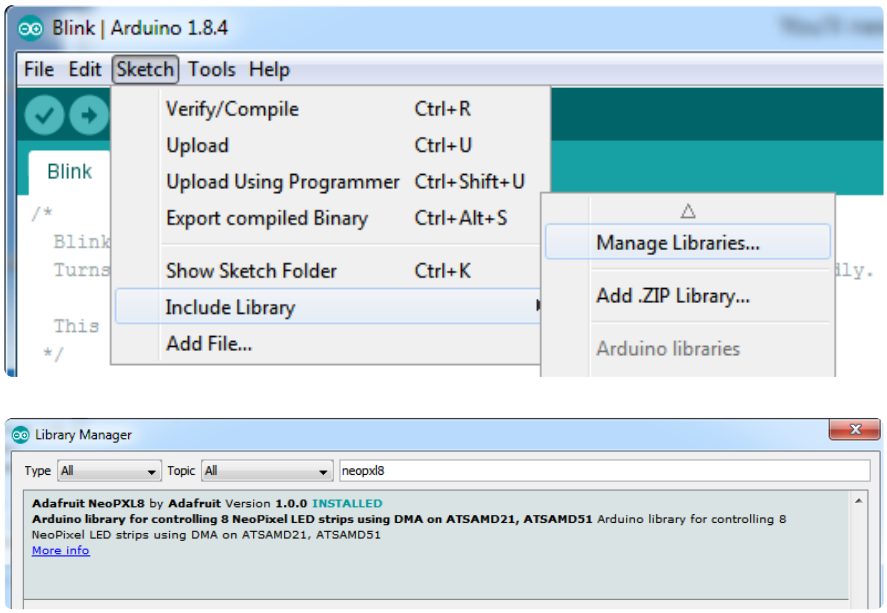
Unlike the FeatherWing, the NeoPXL8 Friend has space for both an 8x2 row header and two RJ45 connectors at the same time.

HOWEVER, if using the 8x2 connector, depending how one’s ribbon cable is crimped and strain-relieved, the RJ45 connectors may still be in the way...so you may want to leave those two connectors off unless you’re certain to need them.

Because we’re using hardware tricks, NeoPXL8 output works only on specific microcontroller pins. The Arduino library examples explain in more detail. On boards like the Adafruit Metro Express or Arduino Zero, it’s conveniently on digital pins 0 through 7, so you can use an 8-pin ribbon cable to link directly to the NeoPXL8 Friend’s 8 inputs. In other situations, the pins may be strewn in different places around the board.

NeoPXL8 Arduino Library

The Adafruit_NeoPXL8 library can be installed using the Arduino Library Manager. You will also need the Adafruit_NeoPixel and Adafruit_ZeroDMA libraries.



Remember to install the correct board support package through the Boards Manager:

- Adafruit SAMD for Adafruit M0 and M4 boards
- Arduino SAMD for Arduino Zero and similar
- Pico/RP2040 Earle F. Philhower III for Feather RP2040 and SCORPIO
- esp32 for Feather ESP32-S3

Example Code

There is a single Adafruit_NeoPXL8 example sketch called “strandtest,” and it’s pretty minimal. There’s no need for a whole graphics demo, because most NeoPXL8 functions are identical to their NeoPixel counterparts, and there are plenty of NeoPixel examples around already. The differences mostly relate to pin assignments.

First, include Adafruit_NeoPXL8.h instead of Adafruit_NeoPixel.h:

```
#include &lt;Adafruit_NeoPXL8.h&gt;
```

Then declare an Adafruit_NeoPXL8 object (instead of an Adafruit_NeoPixel object) — we’ll call our object “leds” in the example — passing up to three arguments:

```
Adafruit_NeoPXL8 leds(LENGTH, PINS, FORMAT);
```

The constructor's first argument is the number of NeoPixels in each of the eight strands. In other words, the total number of NeoPixels will be 8 times this value. If this value is set at 60, then the total number of NeoPixels is 480 (60 × 8).

The strands don't all need to be the same length. In that case, give the length of the l ongest strand. But from the software's point of view there's still 8 times this many pixels...it's going to require that much memory regardless and there will be gaps in how pixels are addressed. It's best and optimal if all 8 strands are in use and the same length, but not required.

NeoPXL8 is a RAM hog, but on most M0 boards, depending on your code's other needs, you can often get 250 RGB pixels per strand (2,000 pixels total). 300 (2,400 total) is starting to push things. Scale back for RGBW pixels, which require about 33% more RAM.

M4 boards have six times the RAM and can handle absurd thousands of pixels...but in reality, you might not want more than a couple thousand, tops, just to allow the CPU enough time to compute all those pixels at a good frame rate. For an M0 project, maybe a few hundred to a thousand or so. There's no hard limit, you just need to experiment what's practical to compute.

The second argument is an 8-element int8_t array indicating which pins to use for outputs 0 through 7.

This is extremely hardware-dependent, and you'll see in the strandtest sketch there are several different arrays given for different situations. For example, using a Feather M0, NeoPXL8 FeatherWing with the default pin assignments and the Fadecandy-style connector, it's:

```
int8_t pins[8] = { PIN_SERIAL1_RX, PIN_SERIAL1_TX, MISO, 13, 5, SDA, A4, A3 };
Adafruit_NeoPXL8 leds(NUM_LED, pins, NEO_GRB);
```

As explained on the prior page, on the Feather M0, outputs 3, 6 and 7 are not negotiable — they must go to pins 13, A4 and A3 (these can be reordered — you can change which is considered output 3, 6 or 7 — but you cannot select completely different pins). For the remaining 5 outputs, there's a limited ability to reassign things:

Output Number	Default Pin	Alternate Pin

0	PIN_SERIAL1_RX	12
1	PIN_SERIAL1_TX	10
2	MISO	11
4	5	SCK
5	SDA	MOSI

Feather M4 is a little different. There, outputs 4 through 7 are not negotiable — they m ust go to pins 13 through 10 (though they can be reordered within that series). For the other four outputs, there's a limited ability to reassign things:

Output Number	Default Pin	Alternate Pin
0	SCK	RX
1	5	TX
2	9	SCL
3	6	SDA

Changing these assignments requires cutting and bridging pads on the component side of the FeatherWing board, plus matching changes to the pins[] array.

If neither pin choice will work for your application, use a value of -1 for that element of the pins[] array, as many as required. You will lose the corresponding NeoPixel output, and it will still take up RAM and pixel indices as if it were there, but the pin can then be used for normal GPIO.

Different boards will have different pinouts, you'll see this in the example code. The Metro M4 uses an entirely different set of pins. And on the Metro M0 (or Arduino Zero), things are super easy...if you want to use digital pins 0-7 as the eight NeoPixel outputs, just pass NULL instead of a pins[] array, or leave this argument off entirely. RP 2040 boards have their own rules, explained on the next page.

The third argument to the constructor is the NeoPixel data format or color order. Different manufacturers of "NeoPixel compatible" LEDs may use a different R/G/B (and sometimes W) byte order...and even among single manufacturers, different production runs may change the order as required for big customers or if they find it can economize the design. This is very similar to the last argument to the Adafruit_NeoPixel constructor, except any NEO_KHZ800 or NEO_KHZ400 values are ignored (only 800 KHz is supported). You can leave this argument off to use the default NEO_GRB color order.

All 8 strands must be the same type and color order; e.g. you cannot mix RGB and RGBW NeoPixels.

From a coding perspective, the rest appears nearly identical to a regular NeoPixel sketch.

Call the object's begin() function to allocate memory and initialize the pin outputs:

```
leds.begin();
```

You can check for a return value of "true" to confirm the allocation was successful.

Then `setPixelColor()` to modify individual pixel values and show() to issue data to the strands, just like a regular NeoPixel sketch.

For functions that take a pixel index (e.g. setPixelColor(), getPixelColor()), all the pixels are treated as if one long continuous strand. For example:

If the strand length was declared as 60...that's 480 pixels total...

- Pixels 0 – 59 are on strand 0
- Pixels 60 – 119 are on strand 1
- Pixels 120 – 179 are on strand 2
- Pixels 180 – 239 are on strand 3
- Pixels 240 – 299 are on strand 4
- Pixels 300 – 359 are on strand 5

- Pixels 360 – 419 are on strand 6
- Pixels 420 – 479 are on strand 7

This is true even if some strands are physically shorter, or if an element in the pins[] array is -1. The unused bits just vanish into the unknown, like that one light switch that doesn't seem to control anything in the house.

One small difference from `Adafruit_NeoPixel` is that the `setBrightness()` function, in combination with `setPixelColor()` and `getPixelColor()`, work better here. The original color value assigned to a pixel using `setPixelColor()` will always be accurately returned by `getPixelColor()`, regardless of the current brightness setting. In `Adafruit_NeoPixel` this is a "destructive" operation and only an approximation is returned.

That's the vital stuff to know.

Remaining functions and esoterica are documented on GitHub. ()

Also maybe helpful: if using other NeoPixel code as a starting point, this blog post () discusses some out-of-favor techniques and how to write more modern NeoPixel code. If it's using the `wheel()` function, it's old!

RP2040 Use

NeoPXL8 now also works on boards with the RP2040 microcontroller, such as the Raspberry Pi Pico () or Adafruit Feather RP2040 ().

By and large it functions the same, with just some minor changes to the rules...

- Requires the Earle Philhower RP2040 board support in Arduino, not Mbed RP2040. If you don't already have this installed, the steps are documented in this guide. ()
- The pin list passed to the constructor...rather than Arduino pin numbers (silkscreened on the top of some boards), this requires GPIO bit numbers, which aren't always the same. On the Feather RP2040 and ItsyBitsy RP2040, GPIO bit numbers are silkscreened on the bottom of the board. On Raspberry Pi Pico, the numbers are the same, and Pico guidelines such as avoiding GP15 apply.

- The 8 GPIO bits must be contiguous, e.g. GP00 through GP07. They do not need to be aligned to any particular boundary though, just contiguous. 1–8, 3–10 and so forth.
- It's more efficient with RAM, requiring about 2X as much as the regular single-strand NeoPixel library, vs. 4X RAM on SAMD chips. And the RP2040 already has lots of RAM!

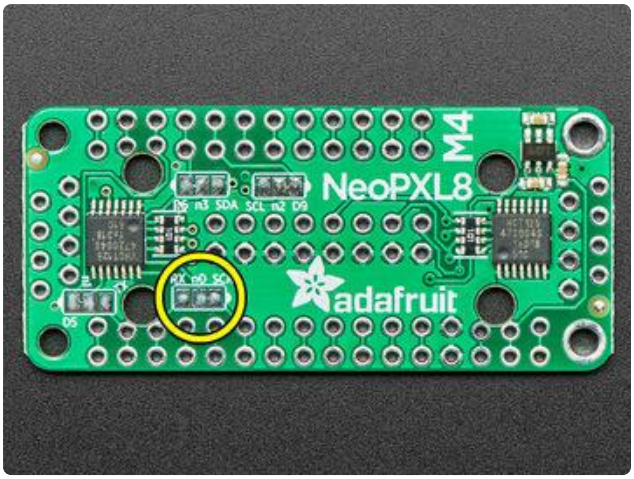
Hardware

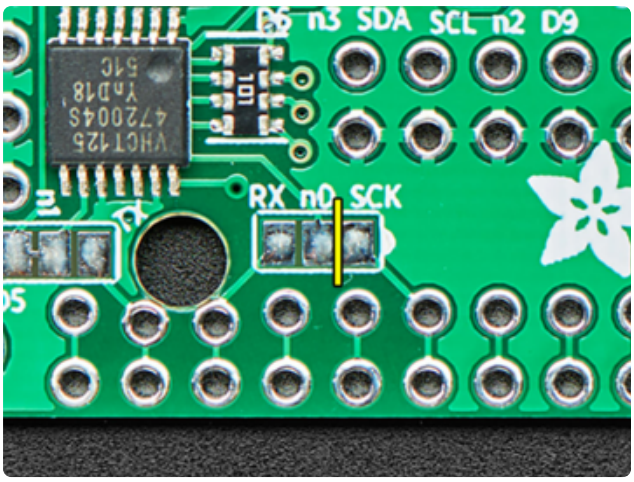
Since RP2040 is a 3.3 Volt device, you'll want to convert logic levels to the NeoPixel supply voltage (5V typ.).

For most boards, our NeoPXL8 Friend () does this nicely. It's explained further on the "NeoPXL8 Breakout Board" page of this guide.

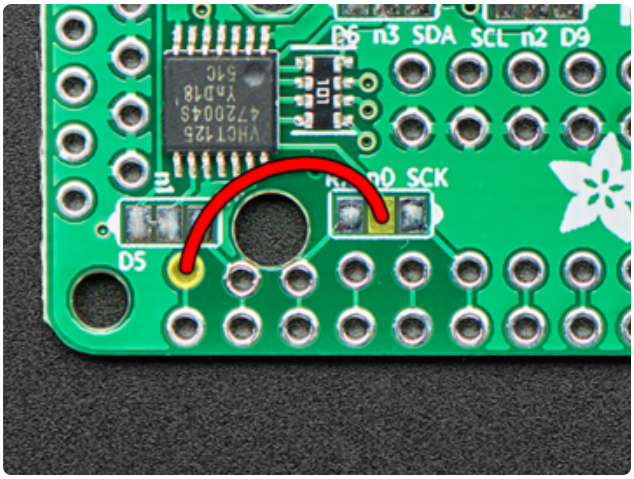
However...if you're using a Feather RP2040 (), and if you're comfortable with some fine soldering, our NeoPXL8 FeatherWing M4 () (not the M0 version) can be adapted. Then you have a tidy package with less wiring!

If the work looks troublesome for your current soldering skill, no worries, you can still wire up a NeoPXL8 Friend. And to reiterate, this modification specifically requires the M4 version of the FeatherWing.



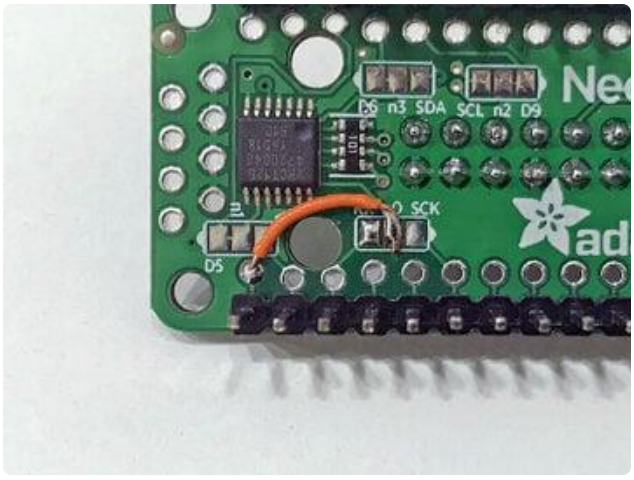


Locate the n0 selector pads on the FeatherWing and use an X-Acto knife or small file to cut the trace between n0 and SCK.



Solder a short piece of insulated solid-core wire between the now-isolated n0 surface pad and the last through-hole via in this row.

If you'll be using the RJ-45 connections for pixels, make sure this wire clears the mounting hole in the board.



This step may take a few tries and some desoldering wick. Those pads are designed for solder bridges, but now we want to keep n0 isolated.

Remember when installing the headers that these components are on the bottom of the FeatherWing.

All other selectable pins — n1 through n3 — should be left in their default configurations.

Software

The strandtest Arduino sketch includes some RP2040-specific notes. You'll need to set up the pins[] list and call the constructor like so:

```
int8_t pins[8] = { 6, 7, 9, 8, 13, 12, 11, 10 }; // GP## indices!
Adafruit_NeoPXL8 leds(NUM_LED, pins, NEO_GRB);
```

On the Feather RP2040, those correspond to digital pins 4, 5, 9, 6, 13, 12, 11, and 10 on the top silkscreen. This is the only 8-pin list that works between NeoPXL8 and the Feather RP2040.

For other hardware, there's usually more flexibility...just provide a list of eight contiguous GPIO bits. They don't need to be in-order, merely contiguous.

And that's it! Most existing NeoPXL8 sketches (which are very similar to regular NeoPixel sketches) will then carry right over. Some may need small changes...and that's not always because of NeoPXL8, it's sometimes unrelated hardware differences.

For example, the Ooze Master 3000 () project can be adapted with just a couple code changes...

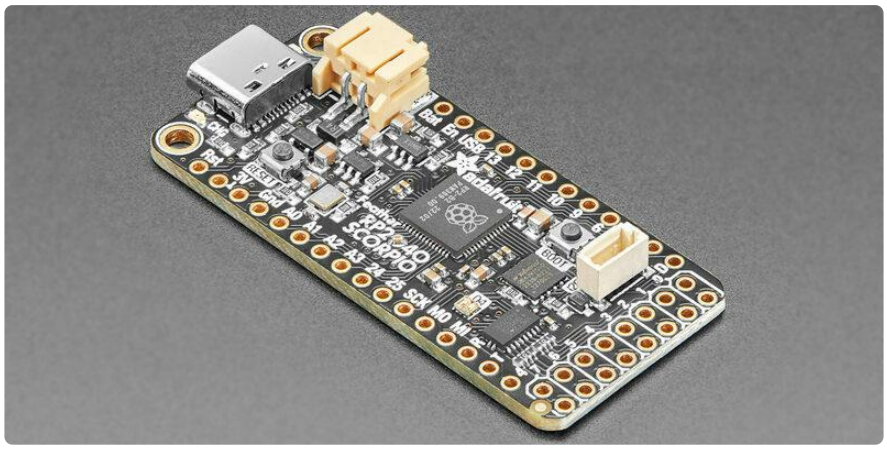
First, the pin list around line 23 should be modified...either the list given above (if using Feather RP2040 + NeoPXL8 FeatherWing M4), or a list of GPIO bits if using a different board with NeoPXL8 Friend.

Second, change the randomSeed() call (around line 68) to reference only pin A0, not A0 + A5. The RP2040 only has four analog inputs. Or you can just delete the line. The code is just using unconnected analog inputs to randomize things on startup, but it's not vital.

```
randomSeed(analogRead(A0));
```

The code should then compile successfully for RP2040 boards.

Feather RP2040 SCORPIO



The board-end 8x2 header on SCORPIO uses a different pin assignment than the regular Feather RP2040. No cutting, no jumpers, these pins have one job:

```
int8_t pins[8] = { 16, 17, 18, 19, 20, 21, 22, 23 };
Adafruit_NeoPXL8 leds(NUM_LED, pins, NEO_GRB);
```

Other than this pin list change, most of what's previously stated on the RP2040 page applies. More informative though, the SCORPIO board has its own dedicated guide ()!

ESP32-S3 Use

NeoPXL8 now also works on boards with the ESP32-S3 microcontroller, such as the Feather ESP32-S3 (). Note that only the S3 chip is supported; original ESP32, S2 and C3 are not compatible.

Hardware

Since ESP32-S3 is a 3.3 Volt device, you'll want to convert logic levels to the NeoPixel supply voltage (5V typ.).

For most boards, our NeoPXL8 Friend () does this nicely. It's explained further on the "NeoPXL8 Breakout Board" page of this guide.

Or...if you're using a Feather ESP32-S3 (), our NeoPXL8 FeatherWings (either the M0 () or M4 () versions) work fine unmodified. Then you have a tidy package with less wiring!

Software

The strandtest Arduino sketch includes some notes regarding pin selection. On the ESP32-S3, any 8 pins can be used for NeoPixel output. If using a NeoPXL8 FeatherWing, use one of the provided pinouts for the M0 or M4 'wing, whichever you've got. And that's it! Most existing NeoPXL8 sketches (which are very similar to regular NeoPixel sketches) will then carry right over. Some may need small changes… and that's not always because of NeoPXL8, it's sometimes unrelated hardware differences.

NeoPXL8HDR

NeoPXL8 addresses the NeoPixel bandwidth issue, improving frame rates for large LED installations and freeing up more processor time for creating "next level" animation. With more potent microcontrollers now in the form of the RP2040 and ESP32-S3, suddenly we can go next-next-level. BEAST MODE!

The NeoPXL8 library includes an extra class called NeoPXL8HDR ("high dynamic range"). You don't need to install an additional library for this, it tags along with the NeoPXL8 installation.

NeoPXL8HDR works a lot like NeoPXL8—in fact the same Arduino sketches can be used as a starting point—but then brings bonus cake:

- Temporal dithering provides more intermediate shades—thousands of brightness levels rather than 256.
- 16-bit color components for nuanced animation; each pixel now allows colors in 48-bit (RGB) or 64-bit (RGBW) formats.
- Gamma correction provides perceptually-linear brightness ramps from the pixels—it's built into the library, you no longer need this in your code.
- Frame-to-frame blending creates smooth transitions even when animation frame rates are low.

NeoPXL8HDR works best with a Feather RP2040 or ESP32-S3 (or other RP2040 or ESP32-S3 board, with suitable level shifting). With two processor cores, one can be dedicated fully to NeoPXL8HDR work. Though it can run on an M4 board, the results are less satisfying (CPU time must be split between user code and the library). Other chips, even the M0 that works so well with "classic" NeoPXL8, are right out, the HDR additions are just too demanding.

Tempering Expectations

NeoPixels still use 8-bit brightness levels, this doesn't magically make them 16-bit. The illusion is done with temporal dithering, quickly alternating the LEDs between two different levels. The effect is noticeable, and one must ride a fine line between ju st noticeable and distracting.

Additionally, even the dithered output isn't fully 16-bit. It produces 12 bits by default, and with shorter pixel runs you might manage 13 bits. This is configurable so you can find a balance between precision and distraction. The important idea is that to your code everything appears 16-bit. The last-step dither reduction is fully handled by the library and does not complicate your task.

If your project doesn't require HDR features, use the much-loved NeoPXL8 class! The HDR class consumes inordinate RAM and resources, and it will not automatically make anything look better; one must specifically code to its strengths. Scrolling text? NeoPXL8. Super smooth plasma flame effects? NeoPXL8HDR.

NeoPXL8HDR relies on frequent refreshes to all the NeoPixel strips. There's an associated bandwidth bottleneck to this, ultimately limiting how many pixels can be controlled while maintaining a sufficient refresh rate. This is subjective and there is no hard limit...but around 1,000 pixels (split 8 ways, and give or take a bit) is a sensible ballpark guideline.

Basic Use

If you've never used either class, read through the NeoPXL8 Arduino Library page first and play with the strandtest example. Get something working on actual hardware and familiarize yourself the concepts and constraints there.

Then, graduating to HDR, let's look at the strandtest_hdr example included with the NeoPXL8 library, comparing it against the simpler non-HDR strandtest. Open both side-by side in the Arduino IDE; we'll just highlight certain sections here.

The two start out very similar. #include the header file, declare a list of pins. Even the constructor accepts the same arguments, it just has a different name for the HDR vs. traditional NeoPXL8 varieties.

```
Adafruit_NeoPXL8HDR leds(NUM_LED, pins, NEO_GRB);
```

Immediately things get strange in the HDR code, where you can see it compiles different code for RP2040 vs. ESP32-S3 vs. M4 (SAMD51) microcontrollers.

A new class function—`refresh()`—is introduced. In NeoPixel or NeoPXL8 code, you call the `show()` function to transmit new color data to the LEDs, done. In NeoPXL8HDR, `show()` hands off new color data to the library, but doesn't actually update the LEDs. That's now the task of `refresh()`, which must be called frequently to perform temporal dithering and all the other niceties.

On RP2040, we put a `refresh()` call inside the `loop1()` function, which runs again and again on the chip's second core, as fast as it can go. This requires installing the Earle Philhower RP2040 package in the Arduino Boards Manager, if you haven't already.

On ESP32-S3, `refresh()` is called in a tight loop in the `loop0()` function, which the `setup()` code pins as a separate task on core 0. Arduino code...the animation logic in this case...always runs on core 1.

On the M4 board, it's necessary to set up a timer interrupt (via the Adafruit_ZeroTimer library) to periodically call `refresh()`. You can see some extra steps needed both above and inside `setup()`.

If you know for a fact that your own project will only ever use one chip or the other, you can trim the extraneous code. If sharing code with others as an open source project, it's more neighborly to support all the different chips.

Once inside `setup()`, each class' `begin()` function is a little different. NeoPXL8 accepts no arguments, it just runs one specific way. NeoPXL8HDR adds some flair:

```
bool status = leds.begin(true, 4, true);
```

The first argument selects whether frame-to-frame blending should be enabled. If you can't generate frequent frames for animation, the library will provide some interpolation for you. This requires an extra 6 (RGB) or 8 (RGBW) bytes of RAM per pixel, atop the library's already voracious needs. This 512-pixel example works fine with it, so it's set `true`.

Second argument establishes the temporal dithering resolution. NeoPixels normally provide 256 brightness levels, or 8 bits. The `4` here bumps this up to 12 bits (effectively, but not actually, the extra being dithered). Long chains of NeoPixels are slow to update and can't use a lot of dithering...so you can dial this down (or up for

short strips) to balance refresh rate with effective color range. Valid range is 0 (no dithering) to 8 (maximum dithering, but probably too infrequent to be useful).

Third argument is whether to enable double-buffering in the underlying NeoPXL8 library, freeing up the CPU to start on the next refresh sooner. Again, uses more RAM, but this 512-pixel demo is not a problem. This argument is currently ignored on SAMD, only the RP2040 provides this, but it's present for code compatibility.

`begin()` returns a status code: `true` if it was able to allocate the required RAM, `false` if not. The example sketch ignores this for brevity because we know it fits, but well-behaved neighborly code may want to respond appropriately (perhaps printing a message to the Serial console and/or blinking a board's built-in LED).

Both examples then call `setBrightness()`, but NeoPXL8HDR works a little differently:

```
leds.setBrightness(65535 / 8, 2.6);
```

The first argument is a peak brightness level from 0 to 65535 (vs. 0–255 in NeoPixel and NeoPXL8). Everything sent to the LEDs will be scaled in proportion, this is just the top value. It's set to `65535 / 8` here (instead of `8191`) just as a fancy and hopefully more readable way of saying "one eighth of the maximum brightness," but the two are equivalent.

Second is a gamma correction factor as a floating-point value. This is a topic extensively covered in other guides () already, but the basic idea is to make "in-between" colors more perceptually linear (because 50% duty cycle doesn't look 50% as bright). A value of `2.6` has worked well for us over the years. It's subjective though, you can try more or less.

The default gamma value if left un-set is 1.0; linear brightness values have a linear effect on duty cycle, which is not perceptually linear. This is on purpose and by design, so that any NeoPixel or NeoPXL8 code brought over to HDR doesn't suddenly yield surprises; it will look more or less the same until you start adapting your code. Notice the NeoPXL8 strandtest code performs its own gamma correction on every pixel, while strandtest_hdr doesn't have to.

The remainder of these two examples—strandtest and strandtest_hdr—is then very similar, with the exception mentioned above that hdr doesn't need to gamma-correct every pixel, it's done automatically. Both produce eight rows of colored "rain," but the HDR version looks better especially among the lower brightness levels. Also they're both using 8-bit color values despite HDR's support for 16-bit. More on that later.

Finer `setBrightness()` Details

First, please, I must reiterate a point from other NeoPixel projects and documentation: `setBrightness()` never was and never will be intended as an animation effect in itself. Configure it once at startup and leave it be…in fact you may get flickering in NeoPXL8HDR if using `setBrightness()` "live." Animated fades should be rendered in code, perhaps with the NeoPixel `fill()` function.

If called with a single argument (no gamma value), `setBrightness()` works just like the original NeoPixel/NeoPXL8 version: the brightness value is 8-bit (0-255) and the library will scale it up. This way, old code will work just as it did before. Only when specifying a gamma value, or in more cases listed below, is the brightness level 16 bits.

"Brightness," in the context of `setBrightness()`, is not a perceptual brightness, but rather a duty cycle. So even if a gamma value is specified, this doesn't apply to the brightness value. Requesting 32767 (or 127 for the 8-bit variant above) yields a peak brightness with about a 50% duty cycle, which will appear more than half as bright. This is normal and by design, again for maintaining "classic" code behavior.

In addition to these two formats:

```
setBrightness(0-255);
setBrightness(0-65535, float gamma);
```

NeoPXL8HDR offers the ability to set peak red, green, blue and white (for RGBW pixels) independently, which can be used to improve color balance (folks often find that NeoPixels appear a bit blue-tinted when all elements are equally lit):

```
setBrightness(red, green, blue);
setBrightness(red, green, blue, gamma);
setBrightness(red, green, blue, white);
setBrightness(red, green, blue, white, gamma);
```

The red, green, blue (and white) values are always 16-bit (0-65535) with these syntaxes; there's no 8-bit mode, no need for back-compatibility since classic NeoPixel doesn't offer this. Gamma is always floating-point, `1.0` is linear, and `2.6` works well in practice.

Another reason for defaulting to linear gamma (rather than imposing a value like 2.6) is that some users may want to provide their own more sophisticated color-correction functions. Leave gamma at 1.0 and then use the 16-bit pixel setting functions…

Setting and Getting Pixel Colors

Your old NeoPixel code will work as before. There's...

```
setPixelColor(pixel, r, g, b);
setPixelColor(pixel, 0xXXXXXXXX);
unsigned long x = getPixel(pixel);
```

...plus the `Color()` and `ColorHSV()` functions for "packing" RGB(W) values and working with HSV colors.

All of these functions work as before and use 8-bit (0-255) brightness values. NeoPXL8HDR will upscale these numbers to the 16-bit range (or decimate back down to 8 bits in the `getPixel()` case).

To work with the full range of 16-bit values, a different set of functions must be used...

```
set16(pixel, r, g, b);
set16(pixel, r, g, b, w);
get16(pixel, &r, &g, &b, &w);
```

In each case, `pixel` is a 16-bit pixel index (0–65535) just like before. For the two set functions, r, g, b (and w if present) are now also 16-bit values (0–65535). There is no "packed" `set16()` function as we don't have 64-bit types just yet.

The rgb(w) arguments to `get16()` are pointers to 16-bit variables (`unsigned short` or `uint16_t`). w can be `NULL` if it's not used, but the others must point to valid destinations.

Adafruit_NeoPixel provides a `getPixels()` function which returns a pointer straight into the buffer where 8-bit colors are stored. High-performance code can bypass `setPixelColor()` and work with the pixel buffer directly, with all the risks that entails.

The Adafruit_NeoPXL8HDR class also provides a `getPixels()` function, but the color values are 16-bit (function returns a `uint16_t *`). One perk is that the values here are always in RGB or RGBW order, no need to reorder for different pixel vintages. Same risks still apply though, like anything with pointers, you can clobber things if you go out of range.

Downloads

Arduino Libraries

- Adafruit_NeoPXL8 code () and documentation () on GitHub.
- Adafruit_NeoPixel code () on GitHub.
- Adafruit_ZeroDMA code () on Github.

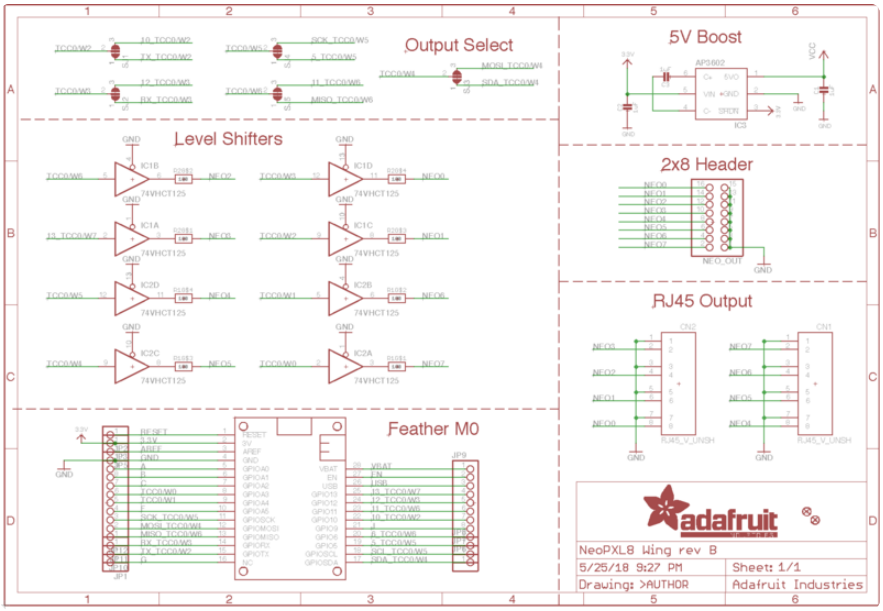
Datasheets and Technical Information

- ATSAMD21 datasheet () (main chip on Feather M0, Metro M0 and Arduino Zero)
- ATSAMD51 information () (main chip on Feather M4, Metro M4 and Grand Central)

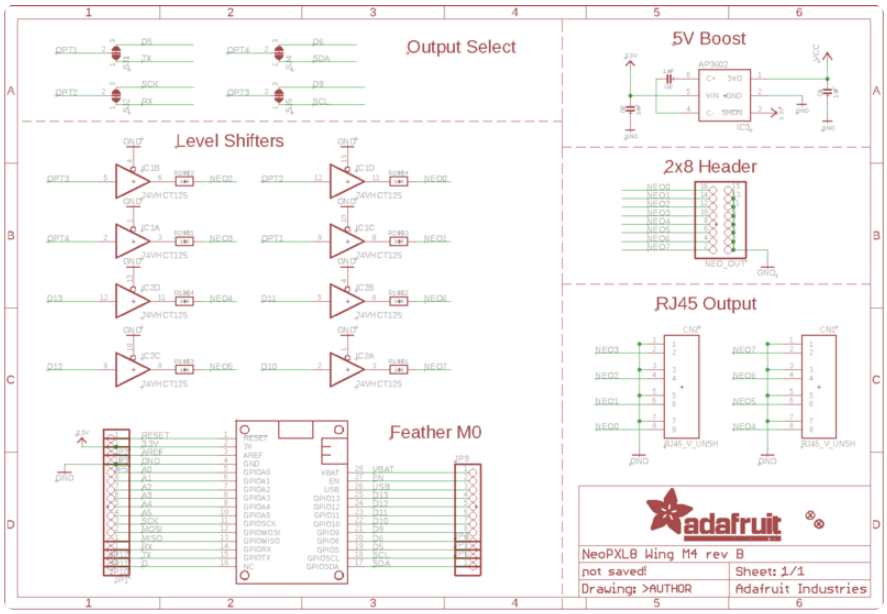
Schematics & Fab Prints

EagleCAD PCB files on GitHub ()

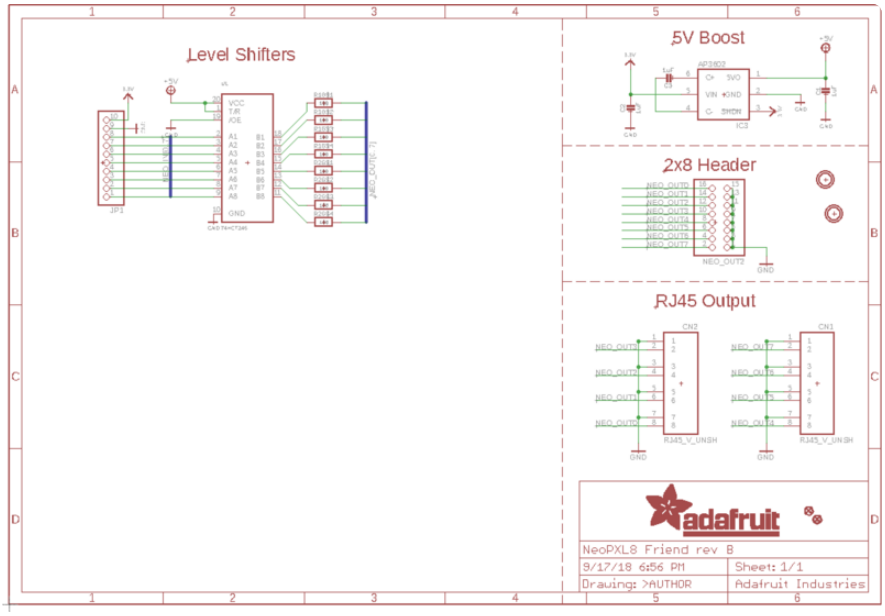
NeoPXL8 FeatherWing M0 (click to embiggen):

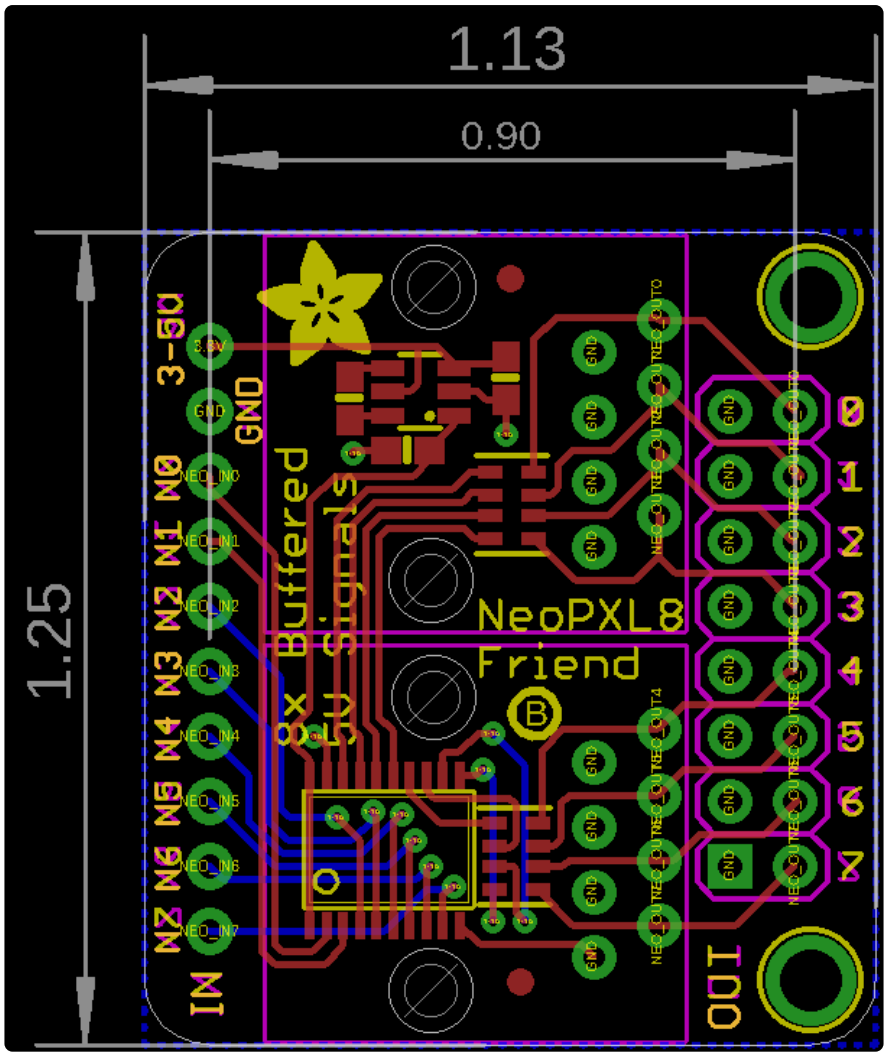


NeoPXL8 FeatherWing M4 (click to embiggen):



NeoPXL8 Friend (click to embiggen):





Components for Fritzing App:

- NeoPXL8 FeatherWing ()
- NeoPXL8 Friend breakout board ()