Adafruit nOOds Überguide

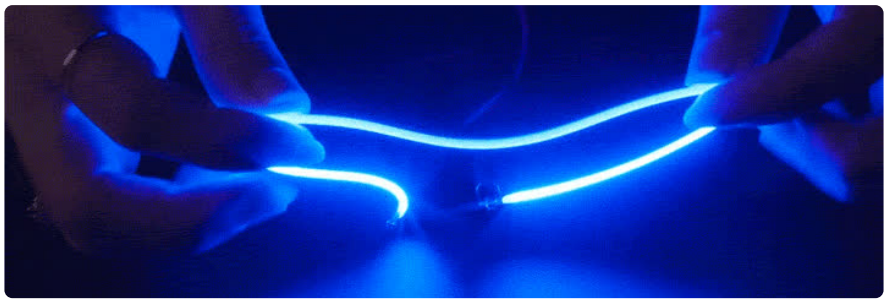
Created by Phillip Burgess

https://learn.adafruit.com/noods-uberguide

Last updated on 2024-06-03 03:41:09 PM EDT

Table of Contents

Overview



Our favorite food when hacking on code or electronics is a hot bowl of noodles — and around NYC these are often called “noods!” What we've got here are flexible LED noodles, in different colors. Not good for eatin' but they are good for cool lighting effects!

These are often seen in “Edison-like” LED bulbs, shaped into hearts or stars or just wound around to create a fun or warm lighting effect. They’re made of dozens of LED diodes that are bonded together on an ultra flexible metal backing, then coated in colorful silicone for protection. Since the LEDs are in parallel, you only need 3V to light ’em up.

Add some mini, noodle-y neon bling to your miniature sets, dioramas, dollhouses, mini-verses, what have you!

"nOOds" is spelled "n capital-O capital-O d s" (no zeros in there).

Differences Between nOOds and EL Wire

There are some **pros** and **cons** compared to electroluminescent (EL) wire **(https://adafru.it/aMg)**...

Pros:

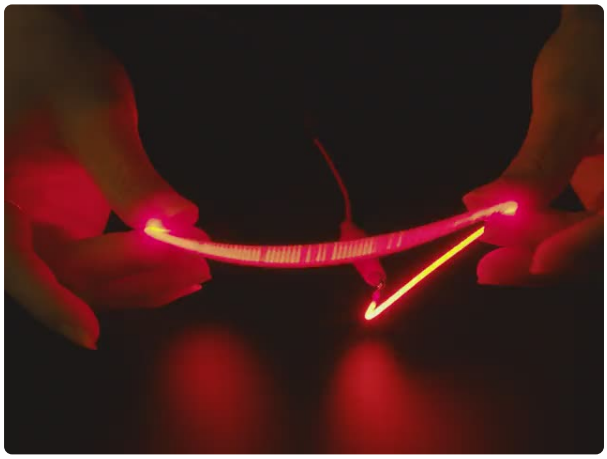
- nOOds run off **regular DC current**. They do not require a special power source (inverter) and do not make any noise.
- nOOds are even **more flexible** than EL wire. They withstand tighter bends (on one axis) and more frequent flexing.
- nOOds can be **dimmed and animated** with pulse width modulation (PWM), from a PWM driver IC or even straight off a microcontroller pin.

Cons:

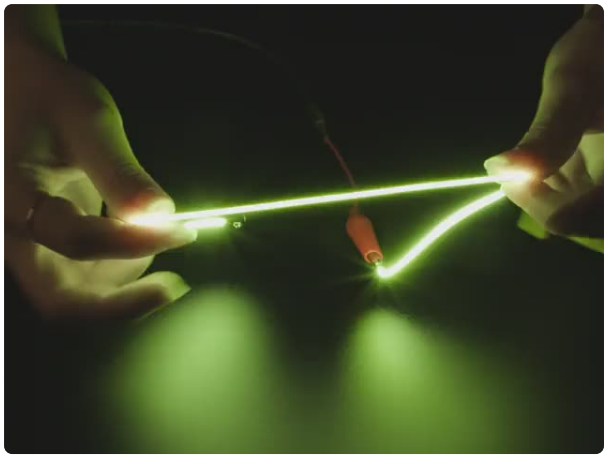
- nOOds **can not be cut** to different lengths. The length they arrive from the factory is what you get, period. The “Tips and Tricks” page offers some ideas on changing the apparent length.
- nOOds require connections at **both ends.** That said, it’s a simple connection, not like the fussy structure of EL wire.

Parts

nOOds - Flexible LED Filament - 3V 300mm long - Warm White
Our favorite food when hacking on code or electronics is a hot bowl of noodles - and around NYC these are often called 'noods'! What we've got here are flexible LED...
https://www.adafruit.com/product/5503



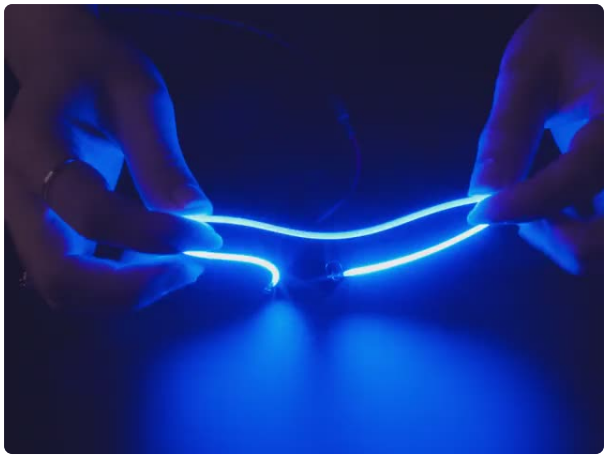
nOOds - Flexible LED Filament - 3V 300mm long - Red
Our favorite food when hacking on code or electronics is a hot bowl of noodles - and around NYC these are often called 'noods'! What we've got here are flexible LED...
https://www.adafruit.com/product/5506



nOOds - Flexible LED Filament - 3V 300mm long - Lime Green

Our favorite food when hacking on code or electronics is a hot bowl of noodles - and around NYC these are often called 'noods'! What we've got here are flexible LED...

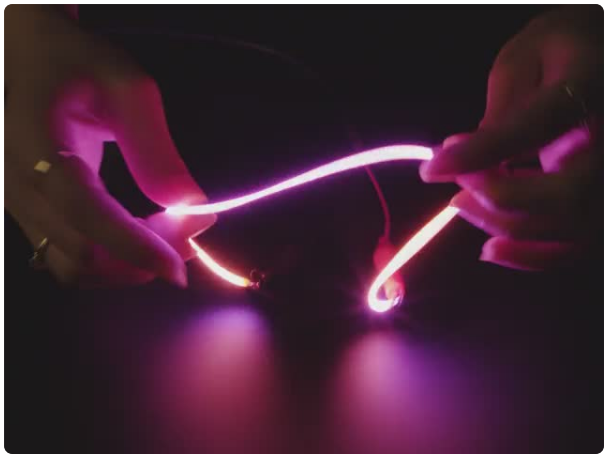
https://www.adafruit.com/product/5507



nOOds - Flexible LED Filament - 3V 300mm long - Blue

Our favorite food when hacking on code or electronics is a hot bowl of noodles - and around NYC these are often called 'noods'! What we've got here are flexible LED...

https://www.adafruit.com/product/5508



nOOds - Flexible LED Filament - 3V 300mm long - Pink

Our favorite food when hacking on code or electronics is a hot bowl of noodles - and around NYC these are often called 'noods'! What we've got here are flexible LED...

https://www.adafruit.com/product/5510

1 x Adafruit AW9523 GPIO Expander and LED Driver Breakout — https://www.adafruit.com/product/4886

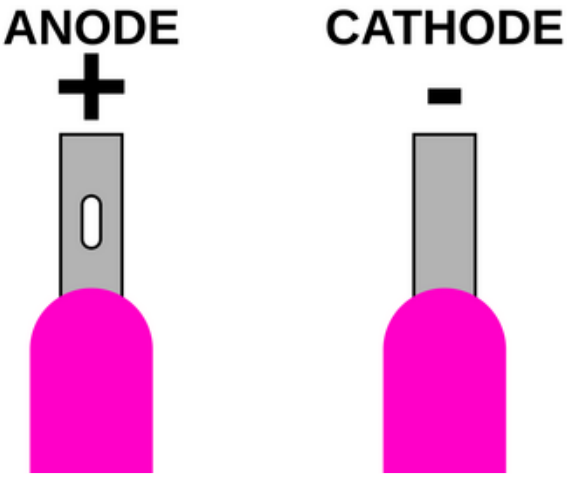
STEMMA QT / Qwiic

---

1 x WS2811 NeoPixel LED Driver Chip — https://www.adafruit.com/product/1378

10 Pack

Electrical Properties



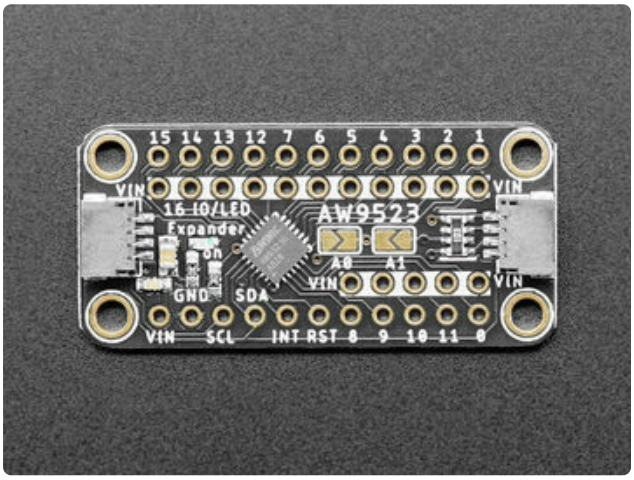
nOOds are comprised of many **light-emitting diodes (LEDs)**, they have a specific **polarity,** with distinct anode ("plus") and cathode ("minus") ends. If a nOOd doesn't light, you might just need to flip it. **The anode end can be identified by a teeny-tiny hole in the metal end tab.**

An inline **current-limiting resistor** is recommended. For 3V nOOds, try around 50 Ohms if the supply voltage is close to 3V, and 220Ω for a 5-ish Volt supplt. For brief tests at these voltages you can probably omit this, but for best longevity it's a smart thing to have.

nOOds can be powered **directly off a 3V coin cell** such as a CR2032. This won't be as bright as with a "proper" power source, but for small items and props it's a great effect and runs for hours...sometimes even a couple days! Because these cells are inherently current-limited, no resistor is needed.

nOOds can be **powered and controlled from microcontroller output pins** via `digitalio` (CircuitPython) or `digitalWrite` (Arduino), and the **brightness modulated and animated** using pulse-width modulation (**PWM**) via `pwmio` (CircuitPython) or `analogWrite` (Arduino). **Here are some things to be aware of:**

- Though most microcontroller GPIO pins are inherently current-limited, it's considered prudent to add a **current-limiting resistor** (as described above) so the chip isn't continually "redlined."
- Every microcontroller has different **current drive capabilities,** with **limits** per pin, per port, and in total. This information will usually be in the "Electrical Specifications" section of the chip datasheet.
- Some microcontrollers can **sink** more current than they can **source.** That is, you might control more and/or brighter nOOds by connecting the cathode (–) end to GPIO pins, and the anode (+) to the microcontroller's voltage, and use inverted logic. Again, check the chip datasheet.
- **Avoid** using `analogio` (CircuitPython) or `analogWrite()` (Arduino) to **DAC-capable pins** (true analog voltage out, not PWM, such as on the SAMD21 A0 pin); LEDs require current control, not voltage control.



Given the vagaries and differences among microcontrollers, rather than controlling nOOds straight off GPIO pins, consider using a **dedicated LED driver** such as the **AW9523** (http://adafru.it/4886). This ensures consistent peak brightness regardless of the type of microcontroller, and dimming is performed via current control rather than PWM; the light is perfectly steady and photographs well. Current-limiting is performed by the device, so no per-nOOd resistor is needed.

nOOds could also be controlled with a WS2811 driver IC (http://adafru.it/1378)— the same logic that's inside **NeoPixels!** This does not make the nOOd per-LED addressable*, but...with three nOOds side-by-side (red, green, blue)...could allow for a sort of color-controllable Neo-nOOd. The WS2811 is a "sink" driver, so the cathode end of each nOOd connects to the IC. The chip provides its own current control (18mA), resistors aren't needed.

* The highest density addressable item Adafruit carries is this half-meter NeoPixel strip (http://adafru.it/4865), but it's much wider and not as flexy as nOOds; not really the same thing.

nOOds can be **connected in series** (end-to-end) with a corresponding increase in voltage, e.g. 3V for one nOOd, 6V for two, 9V for three and so forth. You'll still want a current-limiting resistor. Lower voltages might suffice, e.g. two red nOOds might work from a 5V supply...you'll have to experiment. Probably best and easiest to work with these as parallel, not serial, components.

Physical Properties

Looking closely, you'll see nOOds have a **front face** comprised of a milky white silicone diffuser, and a **back face** that's somewhat transparent. The two faces aren't always perfectly balanced, but close enough for most tasks.

Dimensions

Allow ± a couple percent for normal manufacturing variances, but in general nOOds are...

- **300 millimeters** long from **tip to tip**, including the end connector tabs
- The **illuminated** section is about **285 mm** long
- Exposed portions of end **connector tabs** are about **5 mm** long
- Cross-section is **not perfectly circular**; about **1.7 mm wide, 1.9 mm tall**

Bend Limits

nOOds have an **internal structure**, with distinct **per-axis bend radii.** Think of it like a tiny folding ladder...one axis can fold any which way, the other is unyielding.

In the **front-to-back** direction, nOOds can be fully pinched; the minimum bend radius is equal to the nOOds' radius, about 1 mm. That might be pushing it, but it's possible.

On the **torsional** axis (twisting), nOOds tolerate a full 360° twist about every 25 mm or 1 inch. Less twisting is always better. Too much and you might see individual LEDs pop off inside!

In the **side-to-side** direction...nOOds can't and shouldn't bend! The trick here is to apply a mix of torsion and front-to-back bending. Imagine a banked turn on a racetrack or highway...it's a little like that.

Thus, to achieve the most intricate shapes with the tightest bends, nOOds would ideally be installed sideways. But as explained above, the front and back faces aren't always perfectly balanced in brightness. From any reasonable distance, probably unnoticeable. Tradeoffs!

Durability

nOOds' flexibility makes them a delight to noodle around with. But they're not engineered for infinite noodling. Like any physical thing, they stand a chance of eventually wearing out. We don't know exactly what that limit is or how to characterize it, but it's likely a function of bend radius, flexing duty cycle and some luck.

For **maximum lifespan**, treat these exactly as you would EL wire or flex LED strips: bend them to a shape once and affix them to a solid support.

Realistically, you can probably work these into costumes and other gently-bendable items that see infrequent use (gloves, outerwear), and they might last the lifetime of the item.

If a situation demands frequent, tight flexing, then plan for these to eventually wear out, and design for quick replacement: perhaps pluggable ferrule connectors on the ends, or screw terminals, or just accessible solder points.

Prototyping with nOOds

The **metal tabs** on the ends of nOOds are **too slim** to make good contact with **breadboards**. It might work for a quick test, but for anything more involved will test your patience.

Easiest for quick prototyping is **alligator clips**, such as these gator-to-jumper wires in packs of 6 (http://adafru.it/3448) or 12 (http://adafru.it/3255).

For something better shielded from metal items on your work table, solder **breadboard-friendly wires** onto the ends, apply a little heat-shrink if you like. You can color-code each end for anode vs. cathode!

Attaching nOOds

Here are some ways nOOds might be attached to things:

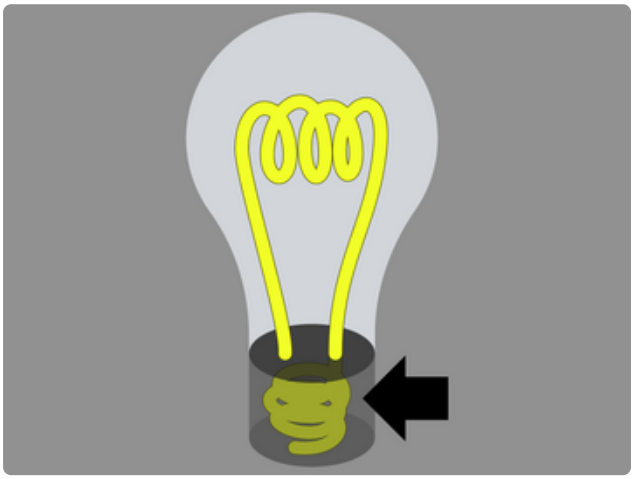
- Monofilament fishing line (e.g. wrapping around wire armature)
- Clear thread (e.g. sewn to garment or to plastic mesh canvas)
- Transparent sticky tape (adhered to flat surface)
- Clear heat-shrink tube (wire armature)

- Press into narrow channel; the nOOds rubbery surface should grip in place (signs and 2D shapes) — a great application for 3D printing or laser cutting!

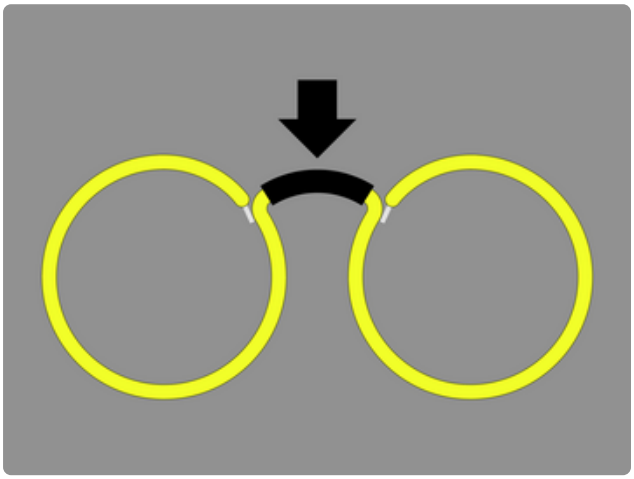
Silicone glues are not currently recommended, as they can be very picky about what sticks to what. Supply chain issues have resulted in some glues being reformulated...a brand that works today might not work the same tomorrow.

Tips & Tricks

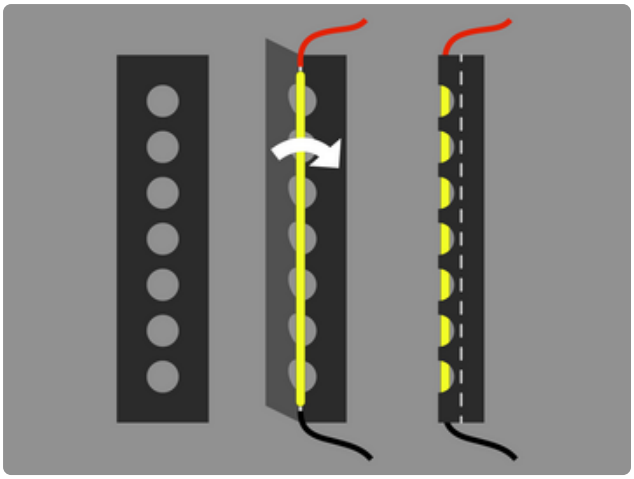
nOOds can not be cut. Period. But that's why this is tips and tricks! Let's do shenanigans...



You can simulate a shorter length by stuffing part of the nOOd behind an opaque base or end piece where it can't be seen.

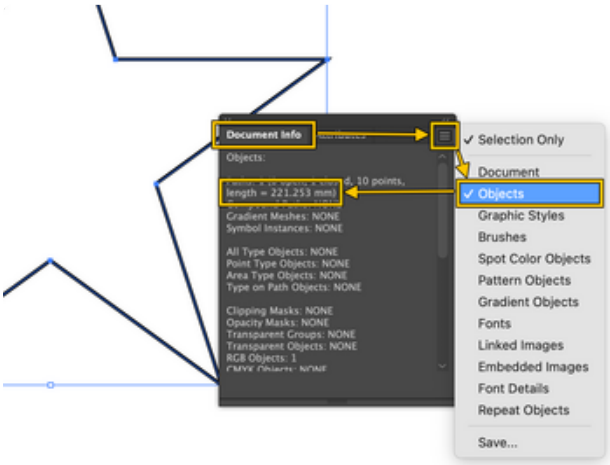


Multiple shorter lengths can be simulated by covering sections of a single nOOd with **opaque black heat-shrink tubing.**



You can create a dotted line effect using punched or laser-cut fabric or leather, folded over the nOOd and stitched flat.

Long nOOds can be simulated by connecting in series (see “Electrical Properties” page). This may result in an unlit gap. You can stagger the nOOds side-by-side near the ends, eliminating the gap but creating a slight discontinuity, or could try pinching the ends back on itself.



If planning designs using **Adobe Illustrator**, and if you want to get **maximum use** of each nOOd (the full length, not having to obscure part of it), here's what to do:

Set document units to **millimeters**.
From the menu bar, select **Window→Document Info**.
From the Document Info window, click the flyout menu in the corner and enable **Objects**. Information about the currently-selected path is now shown, including **path length**.
The ideal path length is **285 mm**, matching the illuminated length of one nOOd. You could **manually scale up or down** until you get close to this, or...
From the menu bar, select **Object→Transform→Scale...** and enter **28500/(current path length)** — for example, with the star path shown here that's 221.253 mm long, you'd type **28500/221.253** (you can enter fractions like that, Illustrator will do the math).
Poof! The path is now exactly **285 mm** long!

nOOds' grippy silicone surface makes them a challenge to feed through tubing such as soda straws. Try the electrician's "fish tape" method: feed a stiff solid-core wire down the tube first, temporarily tack it to one end of the nOOd with a little solder, pull the nOOd into the tube and then de-solder the wire. If there's still too much friction for this operation, rub a few drops of isopropyl alcohol along the nOOd and/or wire before pulling through; it quickly evaporates so there's no trapped moisture.

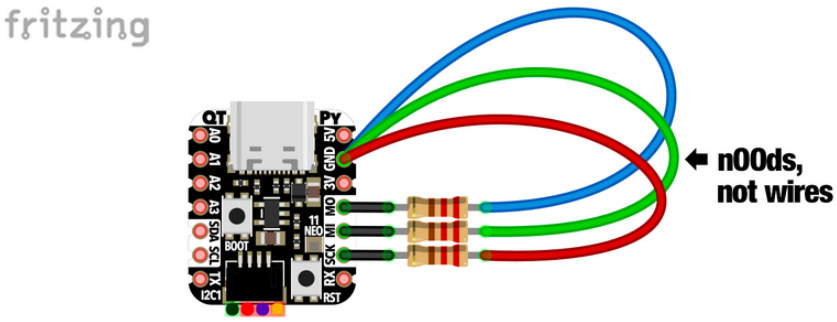
When controlling brightness, whether using PWM or an analog driver like the AW9523, **perceived brightness is not linear**; i.e. 50% duty cycle or 50% current does not look 50% as bright, but somewhat brighter. Just like any other LED project, **gamma correction** (https://adafru.it/w2B) can be applied for a more linear response. Example code on the next page shows this in action.

Example Code

Using GPIO Pins

WIRING: Anode (+) side of nOOds go to GPIO pins, cathode (-) to ground. Current-limiting resistor can go at either end.

This is merely a schematic diagram, you might be using gator clips or a breadboard or whatnot:



Digital Control with CircuitPython

```
# Adafruit nOOds digital control using GPIO pins.
# Uses 3 nOOds, anode (+) to GPIO pin, cathode (-) to ground.
# A current-limiting resistor (e.g. 220 Ohm) can go at either end.

import time
import board
import digitalio

# This uses the 3 adjacent SPI pins on QtPy RP2040, but any pins will do.
PINS = (board.SCK, board.MISO, board.MOSI) # List of pins, one per nOOd

# Convert pin number list to pin object list, initialize to OFF
pin_list = [digitalio.DigitalInOut(pin) for pin in PINS]
for pin in pin_list:
    pin.direction = digitalio.Direction.OUTPUT
    pin.value = False      # All off to start

while True:                # Repeat forever...
    for pin in pin_list:   # For each pin...
        pin.value = True   # nOOd on
        time.sleep(0.5)    # Pause 1/2 sec
        pin.value = False  # nOOd off
```

Digital Control with Arduino

```
// Adafruit nOOds digital control using GPIO pins.
// Uses 3 nOOds, anode (+) to GPIO pin, cathode (-) to ground.
// A current-limiting resistor (e.g. 220 Ohm) can go at either end.
```

```
// This uses the 3 adjacent SPI pins on QtPy RP2040, but any pins will do.
uint8_t pins[] = { SCK, MISO, MOSI }; // List of pins, one per nOOd
#define NUM_PINS (sizeof(pins) / sizeof(pins[0]))

void setup() {
  for (uint8_t i=0; i<NUM_PINS; i++) pinMode(pins[i], OUTPUT);
}

void loop() {
  for (uint8_t i=0; i<NUM_PINS; i++) { // For each pin...
    digitalWrite(pins[i], HIGH);       // nOOd on
    delay(500);                        // Pause 1/2 sec
    digitalWrite(pins[i], LOW);        // nOOd off
  }
}
```

“Analog” (PWM) Control with CircuitPython

```
# Adafruit nOOds "analog" (PWM) brightness control using GPIO.
# Uses 3 nOOds, anode (+) to GPIO pin, cathode (-) to ground.
# A current-limiting resistor (e.g. 220 Ohm) can go at either end.

import math
import time
import board
import pwmio

# This uses the 3 adjacent SPI pins on QtPy RP2040, but any pins will do.
PINS = (board.SCK, board.MISO, board.MOSI) # List of pins, one per nOOd
GAMMA = 2.6  # For perceptually-linear brightness

# Convert pin number list to PWMOut object list
pin_list = [pwmio.PWMOut(pin, frequency=1000, duty_cycle=0) for pin in PINS]

while True:                             # Repeat forever...
    for i, pin in enumerate(pin_list): # For each pin...
        # Calc sine wave, phase offset for each pin, with gamma correction.
        # If using red, green, blue nOOds, you'll get a cycle of hues.
        phase = (time.monotonic() - 2 * i / len(PINS)) * math.pi
        brightness = int((math.sin(phase) + 1.0) * 0.5 ** GAMMA * 65535 + 0.5)
        pin.duty_cycle = brightness
```

“Analog” (PWM) Control with Arduino

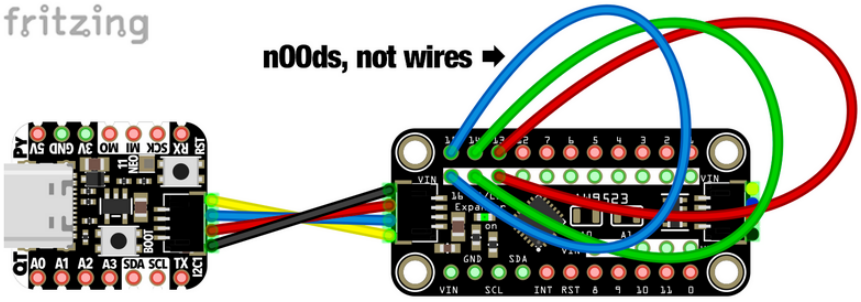
```
// Adafruit nOOds "analog" (PWM) brightness control using GPIO.
// Uses 3 nOOds, anode (+) to GPIO pin, cathode (-) to ground.
// A current-limiting resistor (e.g. 220 Ohm) can go at either end.

// This uses the 3 adjacent SPI pins on QtPy RP2040, but most pins will do.
// Some boards have specific limitations for which pins support PWM!
uint8_t pins[] = { SCK, MISO, MOSI }; // List of pins, one per nOOd
#define GAMMA 2.6                     // For perceptually-linear brightness
#define NUM_PINS (sizeof(pins) / sizeof(pins[0]))

void setup() {
  for (uint8_t i=0; i<NUM_PINS; i++) pinMode(pins[i], OUTPUT);
}

void loop() {
  for (uint8_t i=0; i<NUM_PINS; i++) { // # For each pin...
    // Calc sine wave, phase offset for each pin, with gamma correction.
    // If using red, green, blue nOOds, you'll get a cycle of hues.
```

```
    float phase = (millis() / 1000.0 - 2.0 * i / (float)NUM_PINS) * M_PI;
    int brightness = int(pow((sin(phase) + 1.0) * 0.5, GAMMA) * 255 + 0.5);
    analogWrite(pins[i], brightness);
  }
}
```

Using AW9523 LED Driver

WIRING: Anode (+) side of nOOds go to VIN pins, cathode (-) to numbered breakout pins. Current-limiting resistors are not needed.

This is merely a schematic diagram, you might be using gator clips or a breadboard or whatnot:



IMPORTANT: These code examples are fairly brief. If using AW9523 LED driver as part of a bigger program, **set up the driver and pins as early as possible!** The chip's power-on state does not provide the 18.5 mA current control to LEDs. A tiny blip on startup is harmless, but LEDs shouldn't be kept in this state for any length of time. Related, this is why the examples favor upper pin numbers on the breakout board; these pins provide some current control on power up, though still higher than we'd like (37 mA vs 18.5). Initialization corrects all of this! If this power blip is still a concern, or your code's unable to get to initialization quickly, it's perfectly acceptable to add current-limiting resistors.

Because the AW9523 is a current sink (rather than source), you'll notice when initializing pins, and in the digital control examples, that the **logic is inverted**: `1` or `True` (CircuitPython) or `HIGH` (Arduino) indicates off, `0` or `False` (CircuitPython) or `LOW` (Arduino) is on.

Digital Control with CircuitPython

```
# Adafruit nOOds digital control using AW9523 LED driver breakout.
# Uses 3 nOOds, anode (+) to VIN row, cathode (-) to pins labeled 13-15.

import time
import board
import adafruit_aw9523
```

```
PINS = (13, 14, 15) # List of pins, one per nOOd

# Instantiate AW9523 on STEMMA I2C bus. This was tested on QT Py RP2040.
# Other boards might require board.I2C() instead of board.STEMMA_I2C().
aw = adafruit_aw9523.AW9523(board.STEMMA_I2C())

# Activate pins while converting pin number list to pin object list
pin_list = [aw.get_pin(pin) for pin in PINS]
for pin in pin_list:
    pin.switch_to_output(value=True)  # Initialize pin OFF

while True:                # Repeat forever...
    for pin in pin_list:   # For each pin...
        pin.value = False  # nOOd on
        time.sleep(0.5)    # Pause 1/2 sec
        pin.value = True   # nOOd off
```

Digital Control with Arduino

```
// Adafruit nOOds digital control using AW9523 LED driver breakout.
// Uses 3 nOOds, anode (+) to VIN row, cathode (-) to pins labeled 13-15.

#include <Adafruit_AW9523.h>

uint8_t pins[] = { 13, 14, 15 }; // List of pins, one per nOOd
#define NUM_PINS (sizeof(pins) / sizeof(pins[0]))

Adafruit_AW9523 aw;

void setup() {
  // &Wire1 here refers to the STEMMA connector on QT Py RP2040.
  // On some boards, that might be &Wire or can just be omitted.
  if (!aw.begin(0x58, &Wire1)) {
    Serial.begin();
    while(!Serial);
    Serial.println("AW9523 not found. Check wiring!");
    while (1); // halt
  }
  for (uint8_t i=0; i<NUM_PINS; i++) {
    aw.pinMode(pins[i], OUTPUT);
    aw.digitalWrite(pins[i], HIGH); // nOOd off
  }
}

void loop() {
  for (uint8_t i=0; i<NUM_PINS; i++) { // For each pin...
    aw.digitalWrite(pins[i], LOW);     // nOOd on
    delay(500);                        // Pause 1/2 sec
    aw.digitalWrite(pins[i], HIGH);    // nOOd off
  }
}
```

Analog Control with CircuitPython

```
# Adafruit nOOds analog brightness control using AW9523 LED driver breakout.
# Uses 3 nOOds, anode (+) to VIN row, cathode (-) to pins labeled 13-15.

import math
import time
import board
import adafruit_aw9523
```

```
GAMMA = 2.6         # For perceptually-linear brightness
PINS = (13, 14, 15) # List of pins, one per nOOd

# Instantiate AW9523 on STEMMA I2C bus. This was tested on QT Py RP2040.
# Other boards might require board.I2C() instead of board.STEMMA_I2C().
aw = adafruit_aw9523.AW9523(board.STEMMA_I2C())
for pin in PINS:
    aw.get_pin(pin).switch_to_output(value=True) # Activate pin, initialize OFF
    aw.LED_modes |= 1 << pin                     # Enable constant-current on pin

while True:                         # Repeat forever...
    for i, pin in enumerate(PINS): # For each pin...
        # Calc sine wave, phase offset for each pin, with gamma correction.
        # If using red, green, blue nOOds, you'll get a cycle of hues.
        phase = (time.monotonic() - 2 * i / len(PINS)) * math.pi
        brightness = int((math.sin(phase) + 1.0) * 0.5 ** GAMMA * 255 + 0.5)
        aw.set_constant_current(pin, brightness)
```

Analog Control with Arduino

```
// Adafruit nOOds analog brightness control using AW9523 LED driver breakout.
// Uses 3 nOOds, anode (+) to VIN row, cathode (-) to pins labeled 13-15.

#include <Adafruit_AW9523.h>

uint8_t pins[] = { 13, 14, 15 }; // List of pins, one per nOOd
#define GAMMA 2.6                // For perceptually-linear brightness
#define NUM_PINS (sizeof(pins) / sizeof(pins[0]))

Adafruit_AW9523 aw;

void setup() {
  // &Wire1 here refers to the STEMMA connector on QT Py RP2040.
  // On some boards, that might be &Wire or can just be omitted.
  if (!aw.begin(0x58, &Wire1)) {
    Serial.begin();
    while(!Serial);
    Serial.println("AW9523 not found. Check wiring!");
    while (1); // halt
  }
  for (uint8_t i=0; i<NUM_PINS; i++) {
    aw.pinMode(pins[i], AW9523_LED_MODE);
    aw.analogWrite(pins[i], 0); // nOOd off
  }
}

void loop() {
  for (uint8_t i=0; i<NUM_PINS; i++) { // For each pin...
    // Calc sine wave, phase offset for each pin, with gamma correction.
    // If using red, green, blue nOOds, you'll get a cycle of hues.
    float phase = (millis() / 1000.0 - 2.0 * i / (float)NUM_PINS) * M_PI;
    int brightness = int(pow((sin(phase) + 1.0) * 0.5, GAMMA) * 255 + 0.5);
    aw.analogWrite(pins[i], brightness);
  }
}
```