



# LED Masquerade Masks with n00ds

Created by Erin St Blaine



<https://learn.adafruit.com/led-masquerade-masks-with-n00ds>

Last updated on 2026-03-11 05:25:02 PM UTC

# Table of Contents

<b>Overview</b>	<b>3</b>
<ul style="list-style-type: none"><li>• <a href="#">Parts</a></li></ul>	
<b>Wiring Diagram</b>	<b>6</b>
<ul style="list-style-type: none"><li>• <a href="#">Simple Mask Wiring</a></li><li>• <a href="#">Sequenced n00ds Wiring</a></li></ul>	
<b>Connect one n00d</b>	<b>8</b>
<b>Connect Lots of n00ds</b>	<b>10</b>
<b>CircuitPython</b>	<b>12</b>
<ul style="list-style-type: none"><li>• <a href="#">CircuitPython Quickstart</a></li><li>• <a href="#">Safe Mode</a></li><li>• <a href="#">Flash Resetting UF2</a></li></ul>	
<b>Code</b>	<b>15</b>
<ul style="list-style-type: none"><li>• <a href="#">Upload the Code and Libraries to the RP2040 Prop-Maker Feather</a></li><li>• <a href="#">How the CircuitPython Code Works</a></li></ul>	
<b>Assembly - One n00d</b>	<b>24</b>
<ul style="list-style-type: none"><li>• <a href="#">Inner Glow</a></li></ul>	
<b>Assembly - Lots of n00ds</b>	<b>27</b>
<ul style="list-style-type: none"><li>• <a href="#">Troubleshooting</a></li></ul>	

---

# Overview

Carnival is approaching! In this guide, learn how to add **LED n00ds** to your favorite mask or head piece. n00ds are similar in appearance to EL wire, but don't require bulky or noisy inverters and are a bit simpler to connect. A single 3v n00d runs for hours from a small coin cell battery. They are more flexible than EL wire and come in a variety of lovely colors. Go wild!

This guide will show three different builds:

**Super Simple Inner Glow:** Use a n00d and coin cell battery around the inside of your mask's eye-holes to give your face an otherworldly glow.

**Outlines and Details:** Weave two or three n00ds through the details on your mask to highlight a shape and draw a lot of attention.

**Sequenced Head Piece with Motion Reactivity:** Seven n00ds adorn this glorious headpiece. Learn to sequence or dim the n00ds or have a responsive animation play across them when you tilt your head. This is a more advanced project with a lot of tricky soldering but the possibilities are endless.





guide uses the 3v n00ds at 130mm or 300mm length. The longer n00ds are 12v or 24v and aren't as easy to use for wearables since the battery would need to be huge, or there would be a need to add bulky converters.

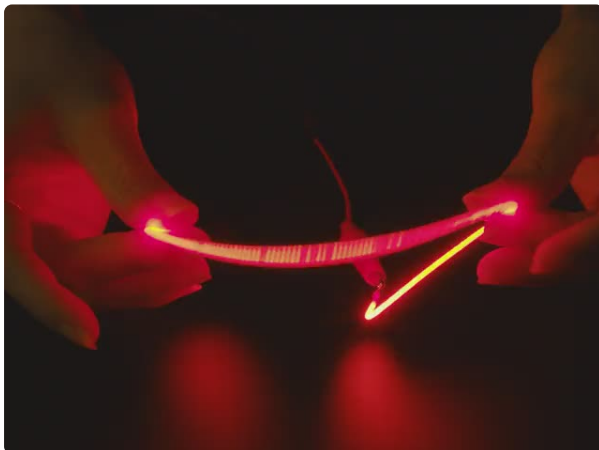
## Parts



### n00ds - Flexible LED Filament - 3V 130mm 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...

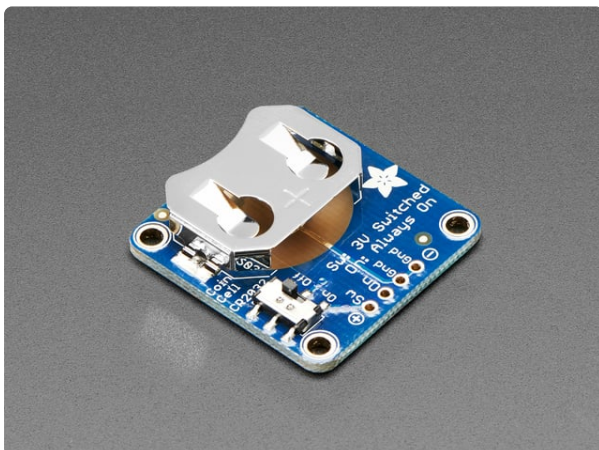
<https://www.adafruit.com/product/5504>



### n00ds - 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>



### 20mm Coin Cell Breakout w/On-Off Switch (CR2032)

Simple but effective - this sewable breakout board has a CR2032 coin cell battery holder soldered on, an on/off switch and 0.1" pitch breakout pins for easy connecting. Great for...

<https://www.adafruit.com/product/1871>

### 3 x CR2032 Batteries

<https://www.adafruit.com/product/654>

CR2032 Lithium Coin Cell Battery

---

### 2 x Silicone Stranded Wire

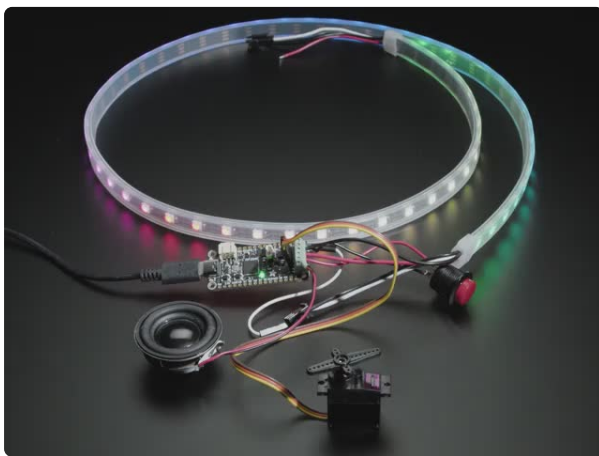
<https://www.adafruit.com/product/>

Silicone Cover Stranded-Core Wire - 2m 30AWG Black

2003

---

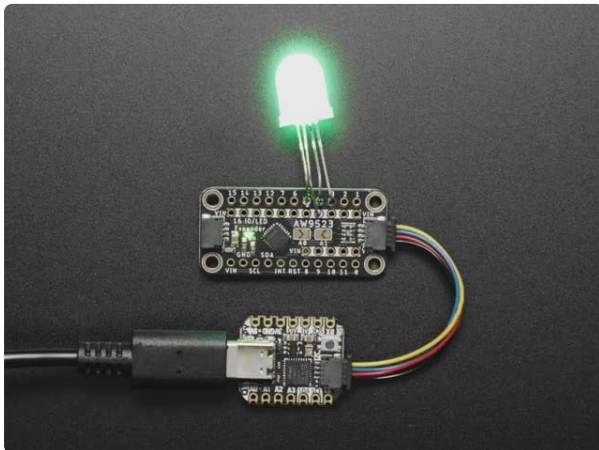
This more advanced Sequenced Antler tutorial is for folks who are ready to take their mask or headpiece to the next level, with lots of n00ds connected to a single lipoly battery. This project will show how to run and sequence the n00ds with CircuitPython code, adding dimming capabilities and sensor reactivity so they pulse and animate, and react to head motions.



### Adafruit RP2040 Prop-Maker Feather with I2S Audio Amplifier

The Adafruit Feather series gives you lots of options for a small, portable, rechargeable microcontroller board. By picking a feather and stacking on a FeatherWing you can create...

<https://www.adafruit.com/product/5768>



### Adafruit AW9523 GPIO Expander and LED Driver Breakout

Expand your project possibilities, with the Adafruit AW9523 GPIO Expander and LED Driver Breakout - a cute and powerful I2C expander with a lot of tricks up its...

<https://www.adafruit.com/product/4886>

### 1 x Stemma Cable

<https://www.adafruit.com/product/4210>

STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long

---

### 1 x 2000 mAh Battery

<https://www.adafruit.com/product/2011>

Lithium Ion Battery - 3.7V 2000mAh

---

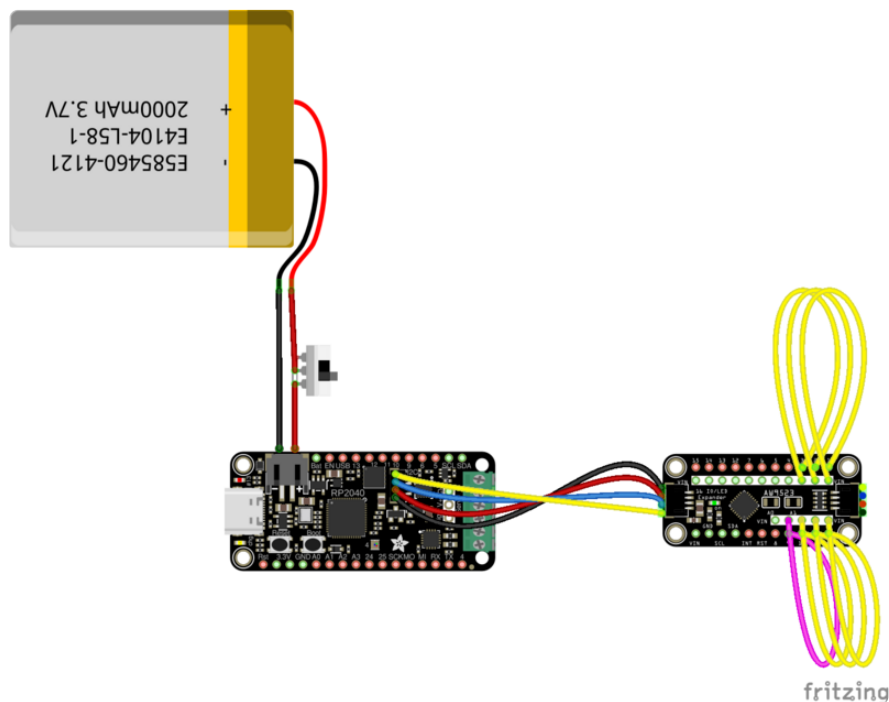


battery for a moment. If it lights up, you've got it right. If not, flip the battery over and try again.

If you're using two or three n00ds on your mask, you'll need two or three battery holders. It's tempting to try connecting two n00ds to a double coin cell holder but you'll get unreliable behavior and flickering.

One battery, one n00d.

## Sequenced n00ds Wiring



This setup uses the AW9523 GPIO expander board with a Feather microcontroller to run up to 16 n00ds from a single battery. I used the Feather RP2040 PropMaker because I wanted motion sensing, but just about any Adafruit Feather controllers will work.

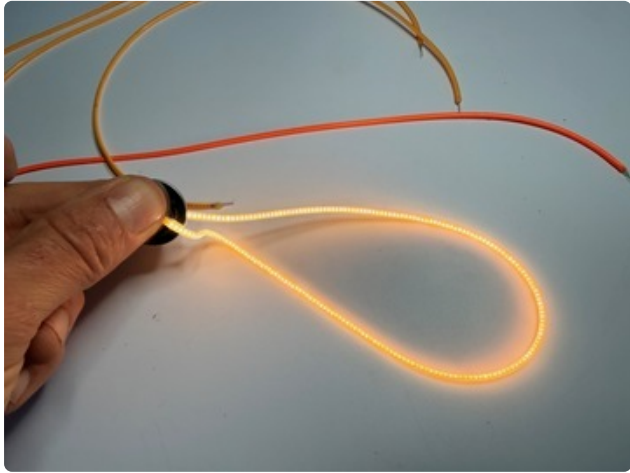
Plug the Feather RP2040 PropMaker into the AW9523 breakout using your Stemma QT cable.

The n00ds can connect to any of the GPIO pins on the AW9523. The + side goes to the VIN pin and the - side to the numbered pin. The sample code uses pins 0, 1, 2, 3, 9, 10, and 11 but this is easy to change to match your setup.

The battery will plug directly into the JST port on the Feather, and the on/off switch will go inline with the red battery cable.

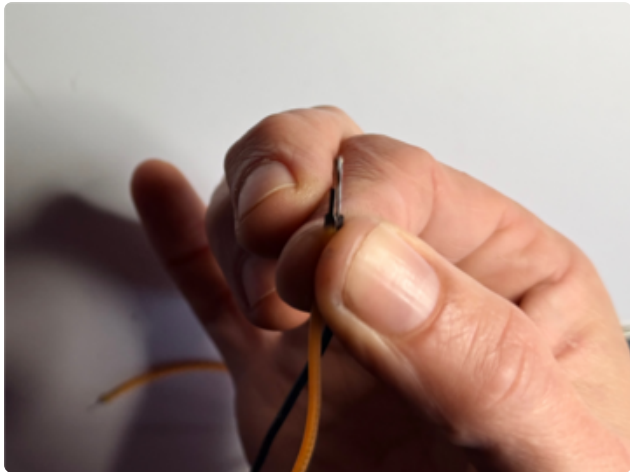
---

## Connect one n00d

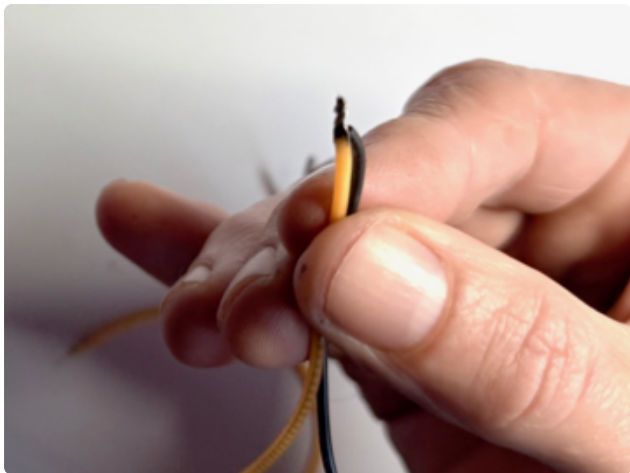


Grab your coin cell battery and press one lead from each end of the n00d to opposite sides of the battery. If it lights up, bingo! The side pressed to the + side of the battery is your + lead. If it doesn't, flip the battery around and see if it lights up when you test it the other way.

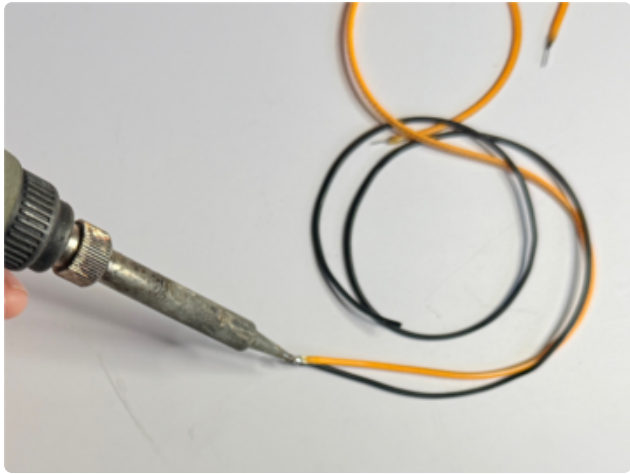
Keep track of which end is + and which end is -.



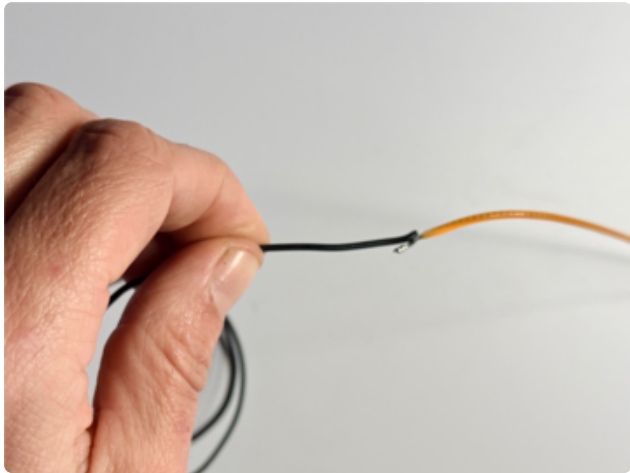
Cut a piece of wire long enough to go from your planned battery location to where you want the n00d to lay out. The wires can be trimmed later, so err on the side of "too long." Strip about 1/4" of shielding from the wire.



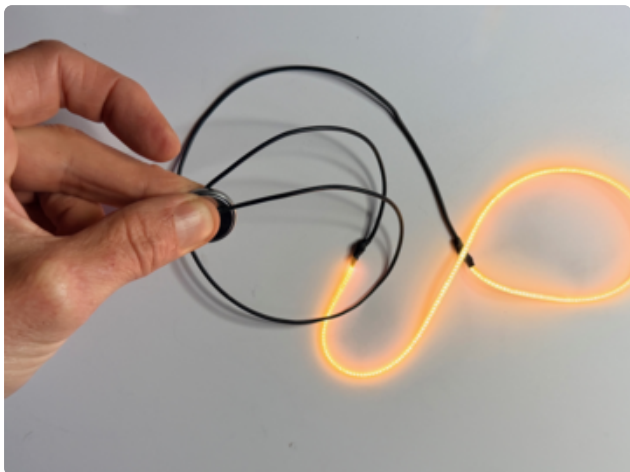
Place the wire and n00d lead next to each other as shown. Twist the bare wire around the n00d lead a few times so they're solidly connected.



Solder the wire and the n00d's lead together and give a tug to make sure you have a solid connection. Pull the wire away from the n00d to get a straight line without bending the n00d's lead -- let all the bend be in the wire. The leads are delicate and you're less likely to break them this way.

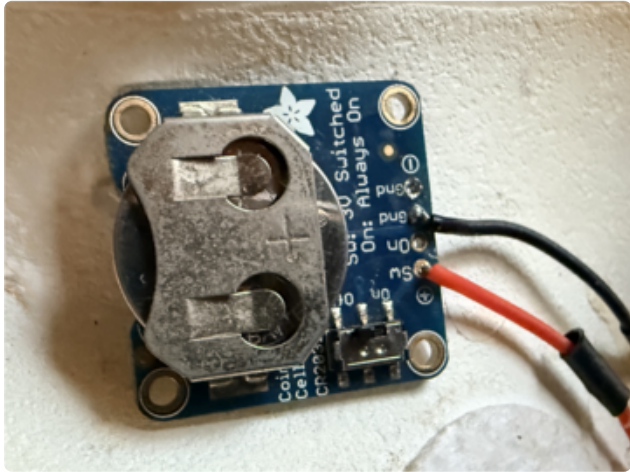


Cover the connection with heat shrink.



Solder another wire to the other end of the n00d. Remember that both ends need to be connected to the power source so make sure your wires are long enough to reach all the way back to the battery.

Test again with your coin cell to make sure it's all working.



The n00d will connect to the coin cell breakout as shown: the + lead (red wire) goes to the Sw pin and the - lead goes to the Gnd pin.

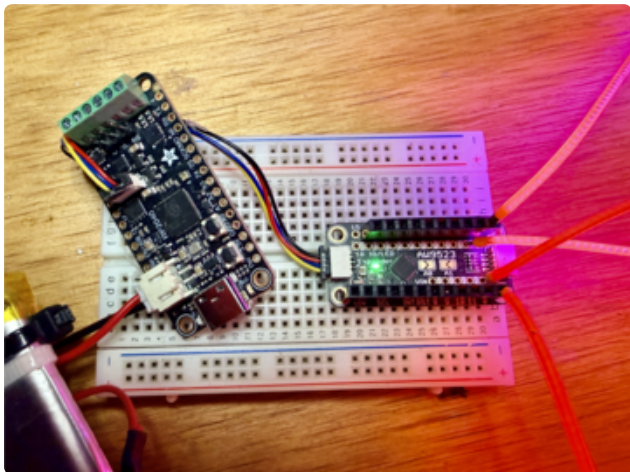
However, if you're planning to weave your n00d into your mask, don't solder it to the battery holder just yet. It's much easier to decorate the mask when you've got the ends of the n00d free, and soldering it all together at the end will allow you to trim the wires to exactly the right length.

---

## Connect Lots of n00ds

The AW9523 GPIO Expander and LED Driver makes it possible to connect a whole bunch of n00ds to your project and drive them from one battery source. It has 16 available GPIO pins and will regulate the voltage automatically, so you don't need to do any calculations or solder a whole bunch of resistors.

[More about the AW9523 GPIO Expander \(https://adafru.it/1aA8\)](https://adafru.it/1aA8)



Plug your Feather RP2040 into the AW9523 using a Stemma QT cable.

The n00ds will be connected to the GPIO pins: the - pad goes into the numbered pin and the + end goes into the corresponding VIN pin on the inner row of the AW9523.

You can connect up to 16 3v n00ds (this won't work with the 12v or 24v), but remember that the more you connect, the larger the battery you'll want to run your project.

The photo shows female headers soldered to the AW9523. I did this for prototyping purposes, but my final build had the wires soldered directly into the pin holes to save space and keep the connections permanent and reliable.

I used a 2000mAh lipoly battery and it runs the 7 n00ds on my project for several hours.

## Order of Operations

1. Decide which GPIO pins you want to use and upload the code to the Feather.
2. Solder extension wires to all your n00ds following the directions on the previous page.
3. Install the n00ds on your headpiece or mask, making sure the extension wires will reach the spot you pick for the controller boards.
4. Connect the extension wires to the AW9523 after the n00ds are all in place and the glue is dry. Remember, the - wire goes to the numbered GPIO pin and the + wire goes to **VIN**.
5. Connect the Feather to the AW9523 using the Stemma QT cable and power it all up.

It feels a little backwards to do it this way, but it will minimize wire trimming and re-soldering if you have the n00ds solidly in place before hooking them up to the controllers.



---

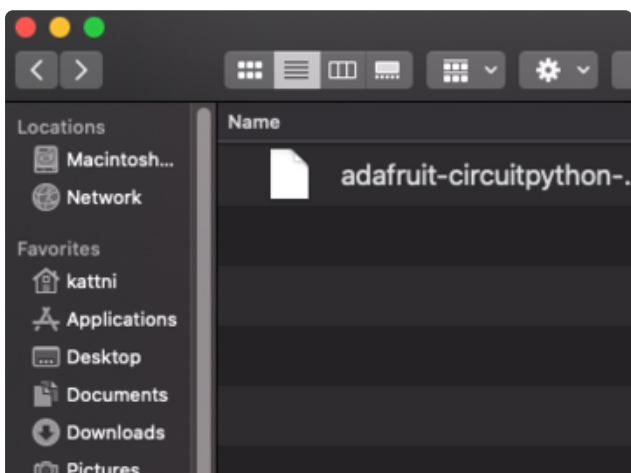
# CircuitPython

[CircuitPython](https://adafru.it/tB7) (<https://adafru.it/tB7>) is a derivative of [MicroPython](https://adafru.it/BeZ) (<https://adafru.it/BeZ>) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

## CircuitPython Quickstart

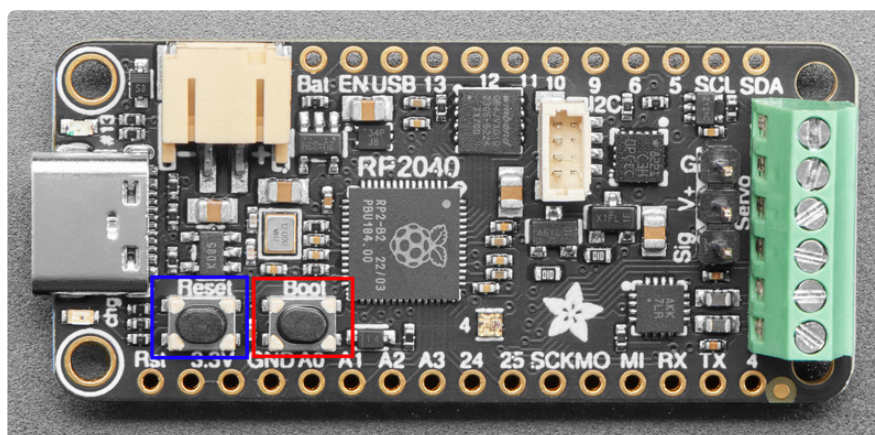
Follow this step-by-step to quickly get CircuitPython running on your board.

<https://adafru.it/18EH>



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

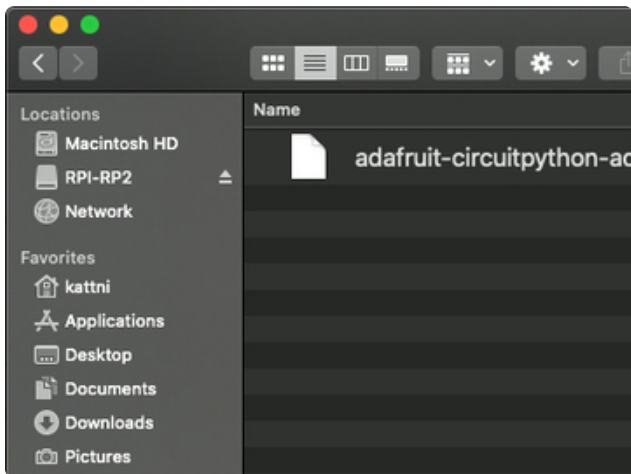


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in red or blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

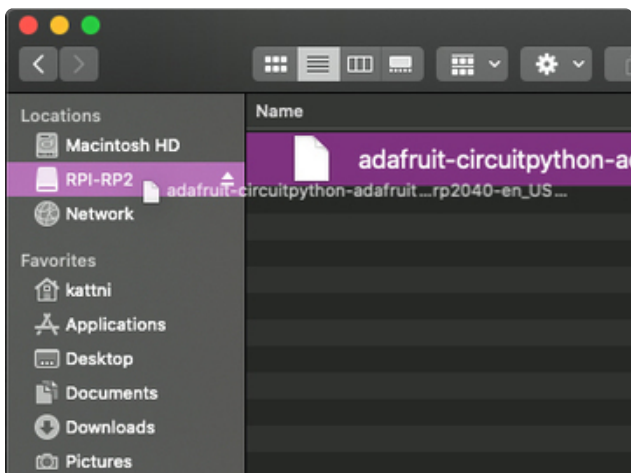
If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

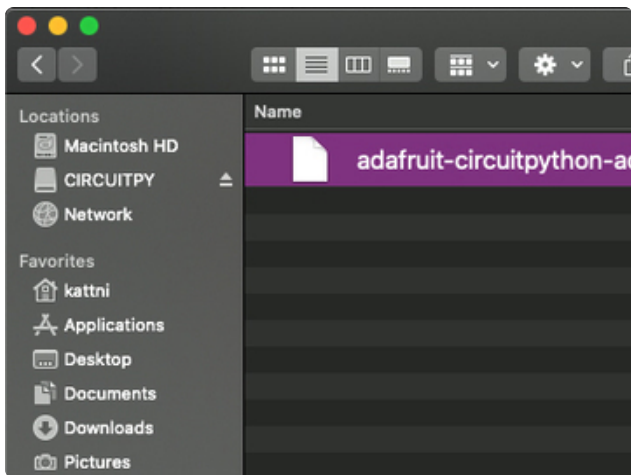
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RPI-RP2**.



Drag the `adafruit_circuitpython_etc.uf2` file to **RPI-RP2**.



The **RPI-RP2** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it, you're done! :)

## Safe Mode

You want to edit your **code.py** or modify the files on your **CIRCUITPY** drive, but find that you can't. Perhaps your board has gotten into a state where **CIRCUITPY** is read-only. You may have turned off the **CIRCUITPY** drive altogether. Whatever the reason, safe mode can help.

Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in **boot.py** (where you can set **CIRCUITPY** read-only or turn it off completely). Second, it does not run the code in **code.py**. And finally, it does not automatically soft-reload when data is written to the **CIRCUITPY** drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the **CIRCUITPY** drive.

## Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

## In Safe Mode

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot.
Press again to exit safe mode.

Press any key to enter the REPL. Use CTRL-D to reload.
```

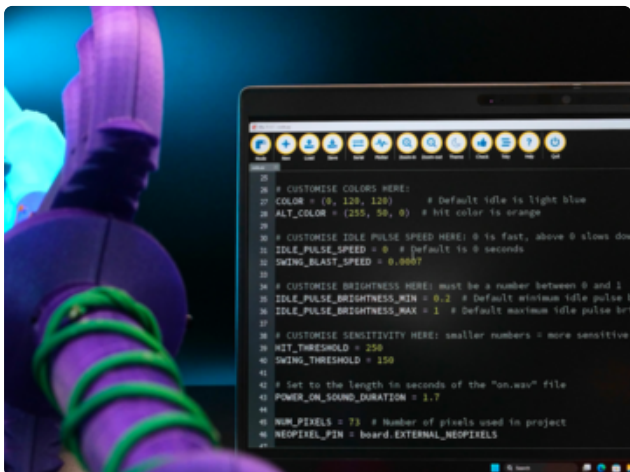
You can now edit the contents of the **CIRCUITPY** drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

## Flash Resetting UF2

If your board ever gets into a really weird state and CIRCUITPY doesn't show up as a disk drive after installing CircuitPython, try loading this 'nuke' UF2 to RPI-RP2. which will do a 'deep clean' on your Flash Memory. **You will lose all the files on the board**, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

<https://adafru.it/RLE>

## Code



Once you've finished setting up your RP2040 Prop-Maker Feather with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the **Download Project Bundle** button in the window below. It will download as a zipped folder.

```
# SPDX-FileCopyrightText: Erin St. Blaine for Adafruit Industries
#
# SPDX-License-Identifier: MIT
"""
```

## Tilt-Triggered Antler Sweep with Idle Breathing Glow

=====

### What this does

-----

This project uses an AW9523 constant-current LED driver to animate 3V LED n00ds arranged as "antlers", plus a center "rose" symbol. A Prop-Maker FeatherWing's LIS3DH accelerometer detects head tilts:

- While you're not tilting: the antlers "breathe" (pulse) between two brightness levels.
- Tilt your head LEFT -> antlers sweep LEFT-to-RIGHT, then back (a "down and back" sweep).
- Tilt your head RIGHT -> antlers sweep RIGHT-to-LEFT, then back.
- A cooldown prevents re-triggering too quickly.

### Hardware

-----

- RP2040 Prop-Maker Feather (LIS3DH accelerometer)
- AW9523 GPIO expander + constant-current LED driver (I2C)
- 7x 3V LED n00ds (wired to AW9523 LED outputs)

### Wiring notes

-----

- AW9523 connects via I2C (STEMMA QT / Qwiic works great).
- Each n00d's + goes to your power rail (3V), and - goes to an AW9523 LED pin.
- The center rose symbol is on AW9523 pin 9 in this example.

### How to tune

-----

All the numbers you'll likely tweak live in the USER SETTINGS section:

- TILT\_AXIS and thresholds: for your board orientation on the headpiece
- BLINK\_OFF\_TIME / BLINK\_GAP\_TIME: sweep feel
- IDLE\_LOW / IDLE\_HIGH / IDLE\_STEP / IDLE\_DELAY: breathing glow speed + depth
- MIN\_SECONDS\_BETWEEN\_TRIGGERS: cooldown

Tip: Start with DEBUG\_PRINT = True, open the Serial Monitor, tilt your head, and watch the axis values. Then adjust thresholds until it feels reliable.

"""

```
import time
```

```
import adafruit_aw9523
import adafruit_lis3dh
import board
import busio
```

```
# =====
# USER SETTINGS (edit these first)
# =====
```

```
# --- Brightness (0-255) ---
```

```
OFF = 0
```

```
DIM = int(255 * 0.60) # Base level used during the sweep (returns to after each blink)
```

```
ROSE = 255 # Rose stays on continuously once it fades in
```

```
# --- Rose fade-in ---
```

```
ROSE_FADE_S = 0.90
```

```
FADE_STEPS = 80 # Higher = smoother fade (slightly more CPU time)
```

```
# --- Sweep timing ---
```

```
BLINK_OFF_TIME = 0.18 # How long each n00d goes dark (OFF) during the sweep
```

```
BLINK_GAP_TIME = 0.04 # How long to wait after returning to DIM before moving to
```

```

the next n00d

# --- Idle "breathing" pulse ---
IDLE_LOW = int(255 * 0.40)
IDLE_HIGH = 255
IDLE_STEP = 3          # Smaller = smoother/slower pulse; larger = faster pulse
IDLE_DELAY = 0.015    # Larger = slower pulse; smaller = faster pulse

# --- Cooldown between triggers ---
MIN_SECONDS_BETWEEN_TRIGGERS = 3.0

# --- Tilt detection ---
TILT_AXIS = "x"        # Change to "y" or "z" depending on how the board is
                        # mounted
TILT_RIGHT_THRESHOLD = +6.0 # Tilt direction "right" if axis value >= this
                        # threshold
TILT_LEFT_THRESHOLD = -6.0 # Tilt direction "left" if axis value <= this
                        # threshold
CONFIRM_SAMPLES = 4    # Require sustained tilt for this many samples

# --- Debug printing (Serial Monitor) ---
DEBUG_PRINT = True
PRINT_EVERY_N_SAMPLES = 2

# =====
# PIN MAPPING (AW9523 pins your n00ds are connected to)
# =====

# Antler groups (by your physical layout)
OUTER = [0, 1]
MIDDLE = [2, 11]
INNER = [3, 10]
ANTLERS = INNER + MIDDLE + OUTER

# Center symbol (rose)
SYMBOL_PIN = 9
SYMBOL = [SYMBOL_PIN]

# List of every AW9523 channel we touch
USED = ANTLERS + SYMBOL

# Sweep order across the antlers (left -> right).
SWEEP_L2R = [1, 2, 3, 10, 11, 0]
SWEEP_R2L = list(reversed(SWEEP_L2R))

def down_and_back(order):
    """
    Convert a one-way sweep list into a "down and back" path without repeating the
    end pin.

    Example:
    [A, B, C, D] -> [A, B, C, D, C, B]
    """
    if len(order) < 2:
        return order
    return order + order[-2:0:-1]

# =====
# HARDWARE SETUP (I2C + devices)
# =====

# Shared I2C bus (STEMMA QT port uses board.SCL/board.SDA)
i2c = busio.I2C(board.SCL, board.SDA)

# AW9523 constant-current LED driver
aw = adafruit_aw9523.AW9523(i2c)
print("Found AW9523")

```

```

# Put all pins into LED (constant-current) mode and configure as outputs
aw.LED_modes = 0xFFFF
aw.directions = 0xFFFF

# LIS3DH accelerometer (Prop-Maker FeatherWing)
lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c)
lis3dh.range = adafruit_lis3dh.RANGE_4_G
print("Found LIS3DH")

# =====
# AW9523 HELPERS (set brightness, fades, and applying updates)
# =====

# Keep track of each channel's current brightness so we can fade smoothly
levels = {p: 0 for p in USED}

def apply_levels():
    """Push our current 'levels' dict out to the AW9523."""
    for pin, value in levels.items():
        aw.set_constant_current(pin, value)

def set_all(value=0):
    """Set all used channels to one brightness value (0-255)."""
    value = max(0, min(255, int(value)))
    for pin in USED:
        levels[pin] = value
    apply_levels()

def set_group(group, value):
    """Set a group of channels (list of pins) to one brightness value (0-255)."""
    value = max(0, min(255, int(value)))
    for pin in group:
        levels[pin] = value
    apply_levels()

def fade_group_to(group, target, seconds):
    """
    Fade a group of channels from their current brightness to 'target' over
    'seconds'.
    """
    target = max(0, min(255, int(target)))
    start = levels[group[0]] # groups stay aligned in this project

    if start == target or seconds <= 0:
        for pin in group:
            levels[pin] = target
        apply_levels()
        return

    dt = seconds / FADE_STEPS
    for step_i in range(FADE_STEPS + 1):
        t = step_i / FADE_STEPS
        v = int(start + (target - start) * t)
        for pin in group:
            levels[pin] = v
        apply_levels()
        time.sleep(dt)

# =====
# IDLE ANIMATION (breathing/pulsing antlers)
# =====

def idle_pulse_step(pulse_dir, value):

```

```

"""
Advance the idle pulse by one small step.

pulse_dir: +1 or -1
value: current brightness value

Returns updated (pulse_dir, value).
"""
value += pulse_dir * IDLE_STEP

if value >= IDLE_HIGH:
    value = IDLE_HIGH
    pulse_dir = -1
elif value <= IDLE_LOW:
    value = IDLE_LOW
    pulse_dir = 1

# Apply pulse to antlers only (rose stays on)
for pin in ANTLERS:
    levels[pin] = value
apply_levels()

return pulse_dir, value

# =====
# SWEEP ANIMATION (blink OFF in a path)
# =====

def blink_off(pin):
    """
    Blink a single antler channel OFF briefly, then return it to DIM.
    """
    levels[pin] = OFF
    apply_levels()
    time.sleep(BLINK_OFF_TIME)

    levels[pin] = DIM
    apply_levels()
    time.sleep(BLINK_GAP_TIME)

def run_sweep_once(order):
    """
    Run exactly one down-and-back sweep (no repeated end pin).

    order: a list of pins that defines the sweep direction (e.g. SWEEP_L2R).
    """
    # Bring all antlers to a consistent base brightness for a clean sweep
    set_group(ANTLERS, DIM)

    path = down_and_back(order)
    if DEBUG_PRINT:
        print("Sweep path:", path)

    for pin in path:
        blink_off(pin)

# =====
# TILT DETECTION (with idle pulsing while waiting)
# =====

def pick_axis(ax, ay, az, which):
    """Return the chosen axis value from the LIS3DH reading."""
    if which == "x":
        return ax
    if which == "y":
        return ay

```

```

return az # "z"

def wait_for_tilt_with_idle_pulse():
    """
    Idle animation + tilt detection in one loop.

    While waiting:
    - antlers pulse between IDLE_LOW and IDLE_HIGH

    Returns:
    "left" if tilted left
    "right" if tilted right
    """
    right_hot = 0
    left_hot = 0
    sample_count = 0

    # Start the idle pulse at the low end
    pulse_dir = 1
    pulse_val = IDLE_LOW

    if DEBUG_PRINT:
        print("\n--- Idling (pulsing) + waiting for tilt ---")

    while True:
        # 1) Do one small idle pulse step
        pulse_dir, pulse_val = idle_pulse_step(pulse_dir, pulse_val)

        # 2) Read accel + evaluate tilt
        ax, ay, az = lis3dh.acceleration
        axis_val = pick_axis(ax, ay, az, TILT_AXIS)

        # Right tilt
        if axis_val >= TILT_RIGHT_THRESHOLD:
            right_hot += 1
        else:
            right_hot = max(0, right_hot - 1)

        # Left tilt
        if axis_val <= TILT_LEFT_THRESHOLD:
            left_hot += 1
        else:
            left_hot = max(0, left_hot - 1)

        # Optional debug printing (Serial Monitor) - f-string to satisfy pylint
        if DEBUG_PRINT and (sample_count % PRINT_EVERY_N_SAMPLES == 0):
            print(
                f"{TILT_AXIS}={axis_val:6.2f} | "
                f"right_hot={right_hot} left_hot={left_hot} | "
                f"idle={pulse_val}"
            )

        # Trigger if the tilt is sustained long enough
        if right_hot >= CONFIRM_SAMPLES:
            if DEBUG_PRINT:
                print(">>> TILT RIGHT DETECTED <<<")
            return "right"

        if left_hot >= CONFIRM_SAMPLES:
            if DEBUG_PRINT:
                print(">>> TILT LEFT DETECTED <<<")
            return "left"

        sample_count += 1
        time.sleep(IDLE_DELAY)

# =====

```

```

# BOOT SEQUENCE
# =====

# Start everything off
set_all(OFF)

# Fade the rose up once, then leave it on
print("Boot: fading rose up...")
fade_group_to(SYMBOL, ROSE, ROSE_FADE_S)
print("Rose is ON.")

# Start antlers at a consistent level before we begin idling
set_group(ANTLERS, DIM)

# Allow an immediate trigger on startup
last_trigger_time = time.monotonic() - MIN_SECONDS_BETWEEN_TRIGGERS

# =====
# MAIN LOOP
# =====

while True:
    # Wait for a tilt while idling (pulsing) in the background
    tilt_direction = wait_for_tilt_with_idle_pulse()

    # Cooldown gate (prevents rapid retriggering)
    now = time.monotonic()
    if (now - last_trigger_time) < MIN_SECONDS_BETWEEN_TRIGGERS:
        if DEBUG_PRINT:
            print("(cooldown) ignoring trigger")
        continue
    last_trigger_time = now

    # NOTE: Your current mapping is intentionally swapped:
    # - tilt_direction == "left" runs the L->R sweep
    # - tilt_direction == "right" runs the R->L sweep
    # This is handy when the board is mounted "flipped" on the headpiece.
    if tilt_direction == "left":
        if DEBUG_PRINT:
            print("\n>>> SWEEP: LEFT -> RIGHT -> BACK <<<")
            run_sweep_once(SWEEP_L2R)

    elif tilt_direction == "right":
        if DEBUG_PRINT:
            print("\n>>> SWEEP: RIGHT -> LEFT -> BACK <<<")
            run_sweep_once(SWEEP_R2L)

    # Return to a known base brightness; the idle pulse will take over immediately.
    set_group(ANTLERS, DIM)

```

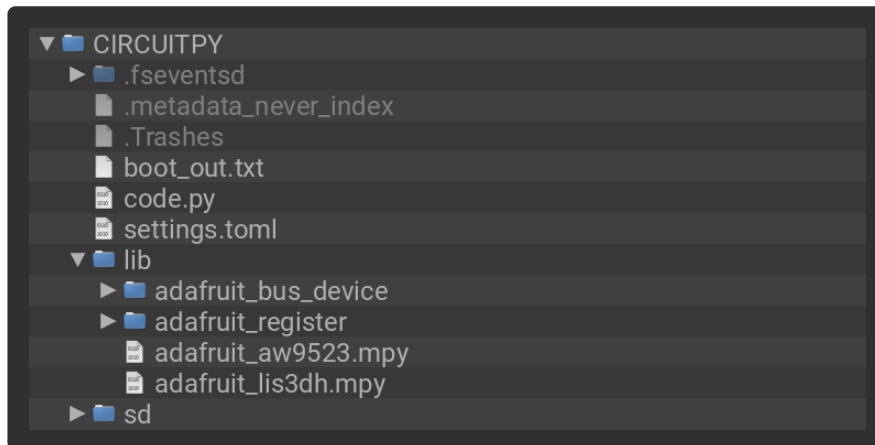
## Upload the Code and Libraries to the RP2040 Prop-Maker Feather

After downloading the Project Bundle, plug your RP2040 Prop-Maker Feather into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called **CIRCUITPY**. Unzip the folder and copy the following items to the RP2040 Prop-Maker Feather's **CIRCUITPY** drive.

- **lib** folder

- **sounds** folder
- **code.py**

Your RP2040 Prop-Maker Feather **CIRCUITPY** drive should look like this after copying the **lib** folder, **sounds** folder and the **code.py** file.



## How the CircuitPython Code Works

The AW9523 drives each n00d as a constant-current LED channel (so brightness is just a 0–255 value), and the LIS3DH accelerometer watches one axis for a sustained tilt. While you’re waiting for a gesture, the antlers do a gentle breathing pulse; when a tilt is detected, the code runs a “down and back” sweep animation across the n00ds.

Start by customizing the **pin mapping** so the code matches your wiring. In the **PIN MAPPING** section you’ll see **OUTER**, **MIDDLE**, **INNER**, and **SYMBOL\_PIN**. These lists are simply the AW9523 channel numbers you soldered your n00ds to. If your build uses a different number of n00ds or a different arrangement, you can rename or remove these groups — the important part is that **ANTLERS** contains every channel you want to animate as antlers, and **SYMBOL\_PIN** points to the center piece (or you can delete the symbol entirely if you’re not using one). Then update the sweep path: **SWEEP\_L2R** is the “physical map” of your antlers from leftmost to rightmost. Put your channels in that order, and the code will automatically create the return path (back toward the start) without blinking the end n00d twice.

```
# =====
# PIN MAPPING (AW9523 pins your n00ds are connected to)
# =====

# Antler groups (by your physical layout)
OUTER = [0, 1]
MIDDLE = [2, 11]
INNER = [3, 10]
ANTLERS = INNER + MIDDLE + OUTER
```

```

# Center symbol (rose)
SYMBOL_PIN = 9
SYMBOL = [SYMBOL_PIN]

# List of every AW9523 channel we touch
USED = ANTLERS + SYMBOL

# Sweep order across the antlers (left -> right).
SWEEP_L2R = [1, 2, 3, 10, 11, 0]
SWEEP_R2L = list(reversed(SWEEP_L2R))

```

Next, tune the **look and feel** using the variables in **USER SETTINGS**. For the idle breathing glow, adjust **IDLE\_LOW** and **IDLE\_HIGH** (the brightness range), and tweak **IDLE\_STEP** and **IDLE\_DELAY** (how fast it breathes). For the sweep animation, **DIM** controls the baseline brightness during the sweep (the level it returns to after each blink), while **BLINK\_OFF\_TIME** controls how long each n00d goes dark and **BLINK\_GAP\_TIME** controls the spacing between steps. If you want the animation to trigger less often (great for a tutorial demo), increase **MIN\_SECONDS\_BETWEEN\_TRIGGERS**. If you want it to feel more responsive, decrease it.

```

# =====
# USER SETTINGS (edit these first)
# =====

# --- Brightness (0-255) ---
OFF = 0
DIM = int(255 * 0.60) # Base level used during the sweep (returns to after each
blink)
ROSE = 255 # Rose stays on continuously once it fades in

# --- Rose fade-in ---
ROSE_FADE_S = 0.90
FADE_STEPS = 80 # Higher = smoother fade (slightly more CPU time)

# --- Sweep timing ---
BLINK_OFF_TIME = 0.18 # How long each n00d goes dark (OFF) during the sweep
BLINK_GAP_TIME = 0.04 # How long to wait after returning to DIM before moving to
the next n00d

# --- Idle "breathing" pulse ---
IDLE_LOW = int(255 * 0.40)
IDLE_HIGH = 255
IDLE_STEP = 3 # Smaller = smoother/slower pulse; larger = faster pulse
IDLE_DELAY = 0.015 # Larger = slower pulse; smaller = faster pulse

# --- Cooldown between triggers ---
MIN_SECONDS_BETWEEN_TRIGGERS = 3.0

```

Finally, make the gesture detection match how your board is mounted. The most common change is **TILT\_AXIS**. If tilting your head doesn't trigger reliably, set **DEBUG\_PRINT = True**, open the Serial Monitor, and try **TILT\_AXIS = "x"**, then **"y"**, then **"z"** while watching which axis value changes the most when you tilt. Once you've picked the correct axis, set **TILT\_RIGHT\_THRESHOLD** and

`TILT_LEFT_THRESHOLD` so a deliberate tilt crosses the threshold, but normal movement doesn't. If it's too sensitive, increase `CONFIRM_SAMPLES` so the tilt must be held for a moment before it triggers. And if the sweep direction is backwards because your board is flipped, swap which sweep runs in the `MAIN LOOP` (the code already includes a note about this and intentionally maps `"left"` to `SWEEP_L2R` in your current version).

```
# --- Tilt detection ---
TILT_AXIS = "x"           # Change to "y" or "z" depending on how the board is
                          # mounted
TILT_RIGHT_THRESHOLD = +6.0 # Tilt direction "right" if axis value >= this
                          # threshold
TILT_LEFT_THRESHOLD = -6.0 # Tilt direction "left" if axis value <= this
                          # threshold
CONFIRM_SAMPLES = 4       # Require sustained tilt for this many samples

# --- Debug printing (Serial Monitor) ---
DEBUG_PRINT = True
PRINT_EVERY_N_SAMPLES = 2
```

---

## Assembly - One n00d

There are lots of ways to attach the n00ds to your mask. You can sew them on using clear fishing line -- the clear thread will disappear when the light shines through. Or if you prefer gluing, use silicone RTV adhesive to secure the n00d. These n00ds are coated in silicone so very few glues will stick to them. Hot glue can work for a while, but most craft glues simply won't grab hold.

I found the silicone glue worked really well, and I also found that weaving the n00d through the metal details on one of my masks worked great too, and allowed me to outline a shape on the mask without having to mess around with sticky glue.

## Inner Glow



Outline the eye holes on the inside of the mask to create an otherworldly glow that makes your eyes into magical portals of wonder.



Use hot glue to secure the battery holder to the mask someplace where it won't rub against your face. I added more hot glue above the wire connections just so my sweat won't short out the solder points.





Weaving the n00d through the mask details can work great too, but it can be a little tricky to get a smooth line. You'll also need to run BOTH ends of the n00d back to the battery pack, so be sure to use a wire color that matches your mask.

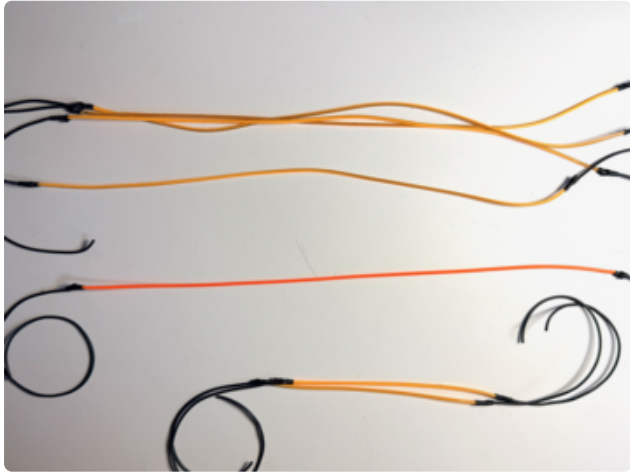


Use one battery holder for each n00d for the best performance.



---

## Assembly - Lots of n00ds



Solder a + and - wire to each n00d. Make them longer than you think you need. It's easy to trim the wires later but tricky and time consuming to extend them.



Figure out your placement and secure the n00ds in place. They're silicone coated, so most glues won't stick but clear RTV silicone sealant works great. It's a little goopy but dries fully clear, and as a bonus, adds some waterproofing too.



I glued the n00ds in place and used zip ties to hold them while the glue dried. It cures to a touchable rubber in about an hour.



Once all the n00ds are in place, manage all the wires. Make sure the wires from both ends of the n00ds all reach back to the spot you're planning to put the controller boards. If any wires look like they might be too short, extend them now.

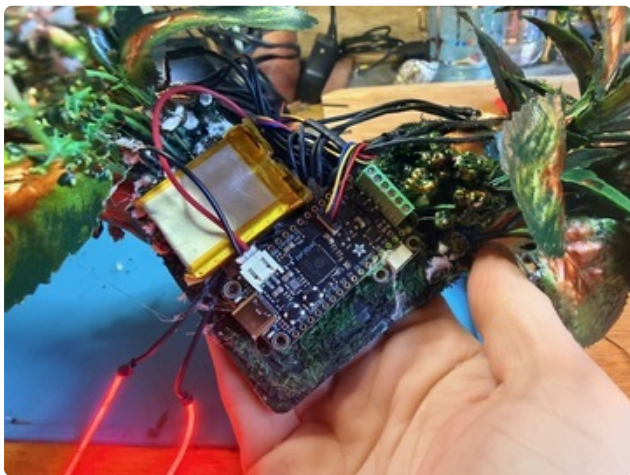


While the glue is drying, add an on/off switch to your battery by cutting the red battery cable and soldering the battery leads to the two halves of the red wire.

Use a zip tie or rubber band around the battery so the wires don't detach from the connection points. These points are delicate and these battery wires pull out easily -- this will save you a lot of heartache.



I used all black wires, which made my life a little difficult at this point -- which wire is which? I figured it out using the coin-cell-battery test trick. Solder the - leads to the numbered GPIO pins, and the + leads to the VIN pins on the AW9523. Take your time and keep track of which n00d is on which pin.



Once the AW9523 is connected to all the n00ds, plug the Feather in using a Stemma QT cable. Plug in your battery and give it a test. If you're using the example code, all the n00ds should come on and pulse slowly. Tilt to the left or right, and the n00ds will animate in sequence.

The way you orient the Feather will determine how well this works. Mine is sitting on top of my head, more-or-less level, with the USB port facing forward. This simple orientation makes the tilt pretty easy to read. If yours just isn't going to work out that way, you can update the code to use the orientation your board requires.

Glue the boards in place with hot glue or RTV silicone.

Keep the battery connector accessible and the battery removable if you want to be able to swap out batteries -- like if you plan to dance in your creation all night long without taking an hour to go recharge. However, it's also an option to bury the battery in the project if the USB port on the Feather is accessible.

This Feather has onboard battery charging capabilities, so you can recharge the battery without removing it from the build, if that works better for your design. Just make sure your battery is switched "on" during the recharge cycle -- the physical switch between the battery and Feather means it can't be charged when it's turned off.



## Troubleshooting

If the n00ds don't turn on at all:

1. Try unplugging and re-seating the stemma cable.
2. Make sure your battery is charged and switched on.
3. Try powering through a USB cable plugged into the Feather -- if they work this way, the issue is with your battery
4. Make sure you can see code.py and the /lib files on the CIRCUITPY drive when you plug the Feather into your computer
5. Make sure the pin definitions are right: the sample code uses 0, 1, 2, 3, 9, 10, 11 -- if you soldered to different pins, fix it in the code.

If some of the n00ds light up but not all, or the sequence is wrong:

1. Check the pin definitions in the code and look closely to make sure you soldered to the pins you thought you did
2. Make sure both ends of the n00d are connected: - to GPIO, and + to VIN. Wiggle the wires and make sure no stray wires are shorting across pins
3. If the sequence is wrong, it's easy to fix in the code - check the previous page for instructions

If the n00ds light up but the motion sensing doesn't work as expected:

1. Try orienting the Feather so it's facing up and the USB port is facing forward -- even just momentarily -- and test again
2. Go into the code and change the tilt axis or the sensitivity until it triggers happily with your preferred motion