3D Printed LED Microphone Flag

Created by Ruiz Brothers

https://learn.adafruit.com/3d-printed-led-microphone-flag

Last updated on 2024-06-03 01:26:10 PM EDT

Table of Contents

Overview

Want to wow the audience at your next gig? Light up those parties with a 3D Printed, LED sound reactive microphone flag. This project uses a microphone sensor to illuminate a custom 3d printed mic flag. You can design yours, or use our STLs to print and make your own!

This guide was written for the 'original' Gemma board, but can be done with either the original or M0 Gemma. We recommend the Gemma M0 as it is easier to use and is more compatible with modern computers!

Parts & Supplies

- Microphone(Omnidirectional)
- Gemma M0 (http://adafru.it/3501) or Gemma v2 (http://adafru.it/1222) (M0 type is recommended!)
- NeoPixel Ring (http://adafru.it/1463)
- Lithium Battery (http://adafru.it/1570)
- Mic Amp [MAX4466] (http://adafru.it/1063) or Mic Amp [MAX9814] (http://adafru.it/1713)either will work
- Double-sided tape
- Solder (http://adafru.it/734)
- 20 gauge wire (http://adafru.it/1311)
- Heat Shrink (http://adafru.it/344)

Tools

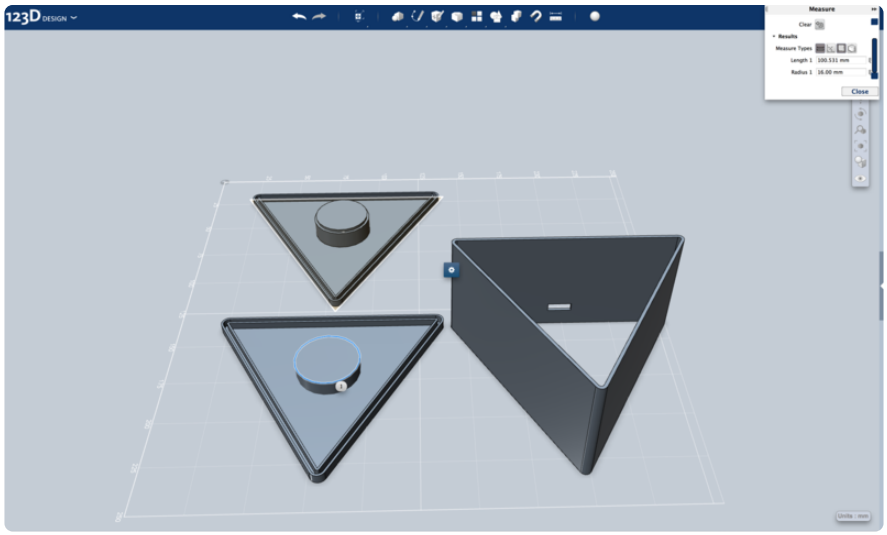
- 3D Printer (http://adafru.it/1292)

- Soldering Iron (http://adafru.it/303)
- Helping Third Hand (http://adafru.it/291)
- Wire cutters (http://adafru.it/527)
- Scissors (http://adafru.it/1599)

Customize

Measure

Our 3D printed enclosure fits a standard microphone with a diameter of 30mm – 34mm. Your microphone may be sized differently, so you'll want to measure the diameter of your microphone using a digital caliper. Since most microphones have are tapered, you should measure the top of the microphone and the center to determine how much area the top and bottom covers of the mic flag need to be.



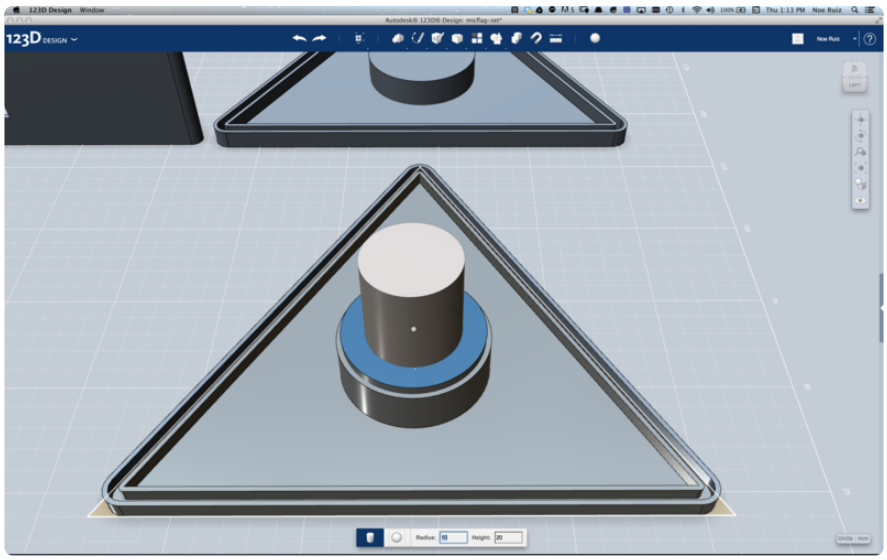
3D Modeling

We have created the parts in 123D Design in a way that allows you to customize the hole for the microphone. The three pieced design is setup so you can easily add a cylinder and cut it out of the two covers. The bottom and top covers are designed to snap tightly onto the frame. The frame has a cutout that fits the slide switch.

Bottom Cover

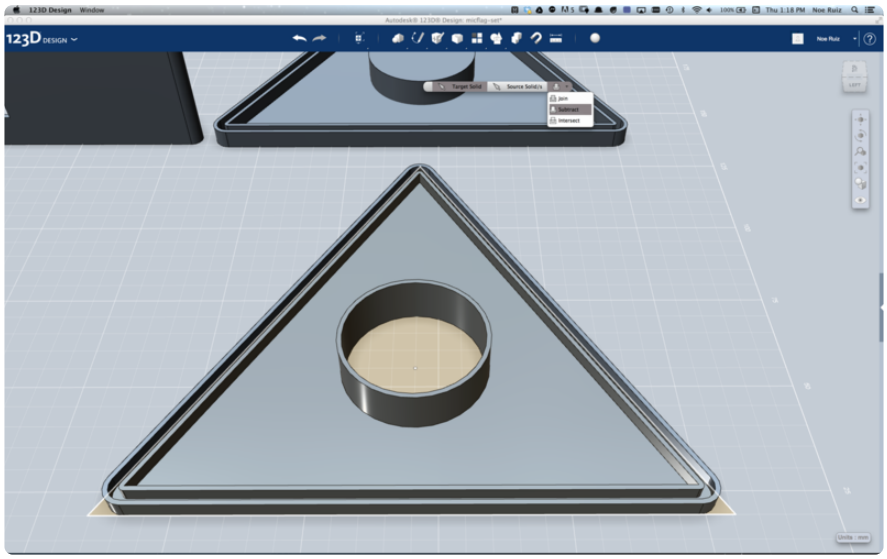
This part is were the circuits will reside. You'll notice this piece is has a smaller diameter and it has a pipe extruded on the edge of the cut out. This is so that the NeoPixel ring can rest on the pipe. The thickness of the pipe should be 1mm thick so that it can grip onto the NeoPixel Ring.

To add a new diameter to the cover, you can add a new cylinder object to the top of the existing cylinder. In the top file menu, under primitives select the cylinder icon. Move your mouse curser over the center of the existing cylinder, it should automatically snap to the center. Use the bottom options menu to change the diameter of the cylinder. The height of the cylinder should be at least 2mm (default 20mm is fine).



Combine + Subtract

You will need to subtract the cylinder to the cover so that it cuts the whole into the center. In the top file menu, select the combine icon and switch the option from Join to Subtract under the drop down menu. Select the bottom cover first, and then click on the cylinder second to highlight it. Press enter or click on an empty area on the grid to accept and confirm your cut out.



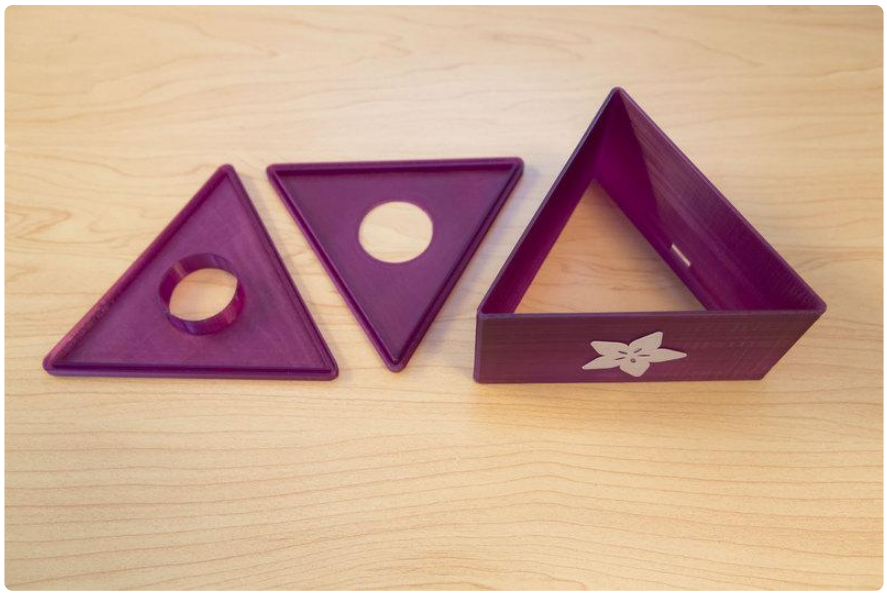
Save + Export

You can now save and export your parts as an STL file. You can find the export STL option under the 123D Design file menu and selecting export STL. We recommend printing each piece individually so that you minimize the changes of a failed print (If you print out a set and something goes wrong, all your pieces will go bad, and thats no bueno!). To save out each part out of 123D, you can temporally delete the parts and leave one to export the STL individually outside of the set. Just remember to undo (cmd/cntrl+Z that baby!) after the export. Repeat for each part. Now onto slicing!

3D Printing

Transparent PLA

We recommend using a transparent PLA material for the frame. This material is great for diffusing the LED lights from the NeoPixel ring. There are color options so you can get creative to match your design.



Download STLs

https://adafru.it/d2v

Printing Techniques

Build Plate Preparations

There's a great video tutorial (https://adafru.it/cRd) by Dr. Henry Thomas who demonstrations a great technique for preparing acrylic build plates for awesome prints. Wipe down the plate with a paper towel lightly dabbed in acetone. Use another paper towel and apply a tiny dab of olive oil. Wipe down the plate so a small film of oil is applied, this will allow the parts to come off the plate easier.

Live Level

We recommend going raft-less for each piece because it will have the best quality result. Each piece will require a well leveled platform. We tend to "live level" our prints, meaning we adjust the build plates thumb screws while the print is laying down filament. This way we can make adjustments directly and improve the leveling by seeing how the extruders are laying down the first layer onto the build plate. We recommend watching the first layer so that you get a more successful print. If you see the layers aren't sticking or getting knocked off, you can always cancel print, peel it off and try again.

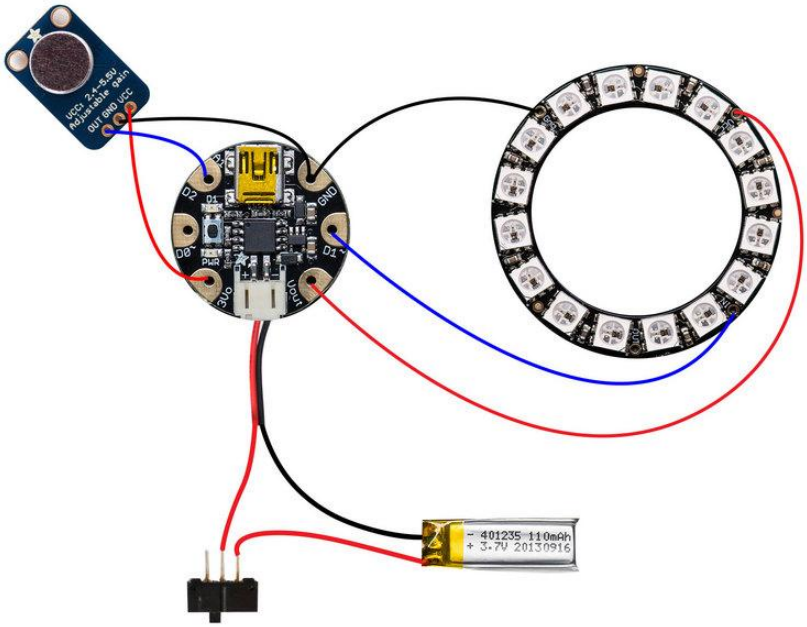
Frame About 3 hours 35g	PLA @230 No Raft No Support	2.0 Layer Height 90/150mm/s

Bottom cover About 55 minutes 10g	PLA @230 No Raft No Support	2.0 Layer Height 90/150mm/s
Top cover About 50 minutes 9g	PLA @230 No Raft No Support	2.0 Layer Height 90/150mm/s

Test for Tolerances

You will need to test your modifications to see if the bottom and top covers tightly fit onto your mic. Since each mic is different, you may need to do several tests before getting the tolerances right. Start by pushing your hand-held microphone through the bottom cover (the one with the smaller hole cut out and the extruded pipe). It should tightly fit onto the mic. Next, position the frame on top and snap it into the bottom cover. Now place the top cover over the mic. We had to remove our mic filter (the wired mesh was removable by unscrewing it) in order to get the top cover on. The top cover should rest on the ledge where the mic filter resides. The height of the frame, or the distance between the bottom and top cover should be enough to close the enclosure.

Circuit Diagram

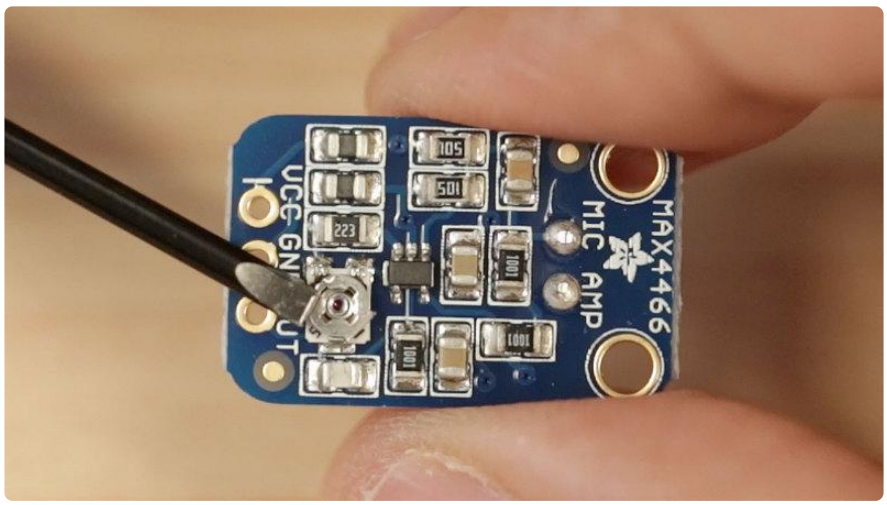


This diagram uses the original Gemma but you can also use the Gemma M0 with the exact same wiring!

Prototype Circuit

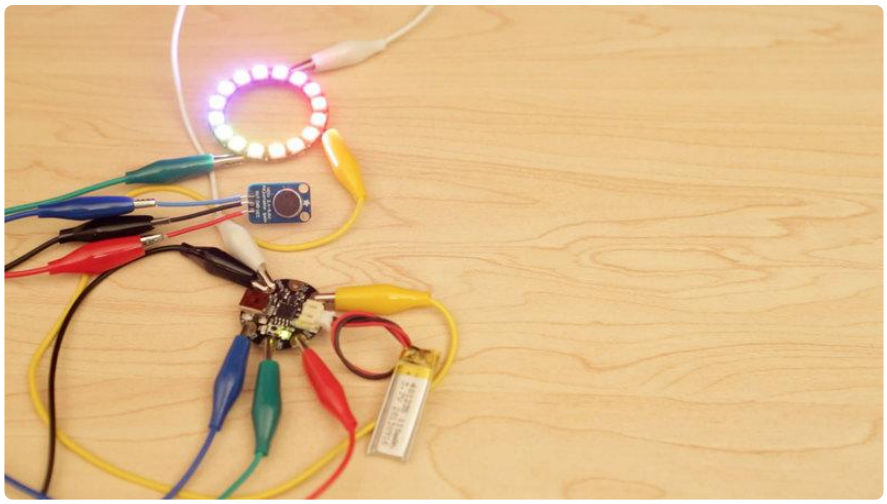
The circuit above illustrates how to connect the components together. You should prototype the circuit before soldering wires to the Gemma. The NeoPixel ring and Mic sensor will be connected to the Gemma. Connect three alligator clips to the NeoPixel Ring pins labeled, **GND**, **DIN**, and **5V**. These three alligator clips need to connect to the Gemma. **GND** on of the NeoPixel connects to **GND** pin of the Gemma. **DIN** pin of the NeoPixel connects to the **D1** pin of the Gemma. **5V** on the NeoPixel connects to the **vbat** pin of the Gemma.

The mic sensor will also need three alligator clips. **VCC** of the mic sensor goes to **3v** on the Gemma. The **out** pin on the mic goes to **A1/D2** on the Gemma. **GND** of the mic sensor will shares the **GND** connected to the Gemma. The mic has an adjustable gain. On the back of the mic PCB, there's a tiny screw that you can adjust to lower or increase the sensitivity of the input gain.



Trouble Shooting

Please take consideration when using alligator clips. If the circuit isn't working, double check the connections on the NeoPixel ring and the mic sensor. They have tiny pins and can often be difficult to alligator clip. If the prototype is too flakey, try soldering jumper wires to the pins of the NeoPixel ring and the mic sensor and then alligator clipping them to the Gemma.

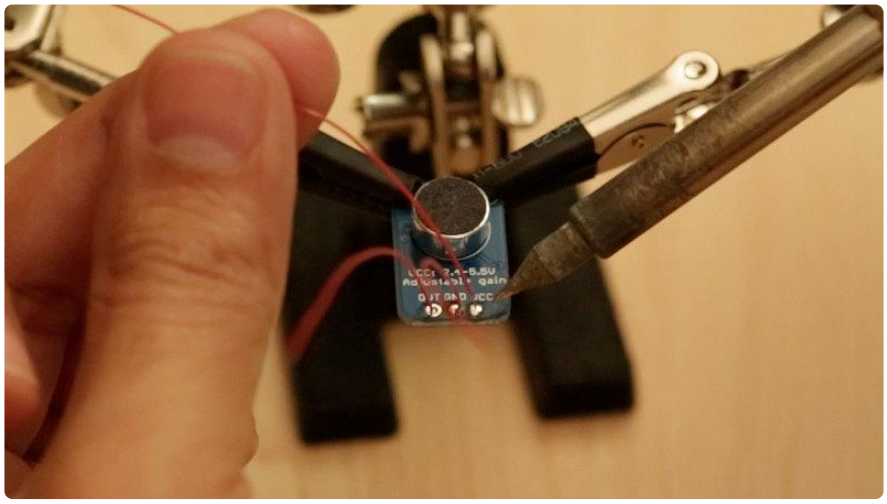


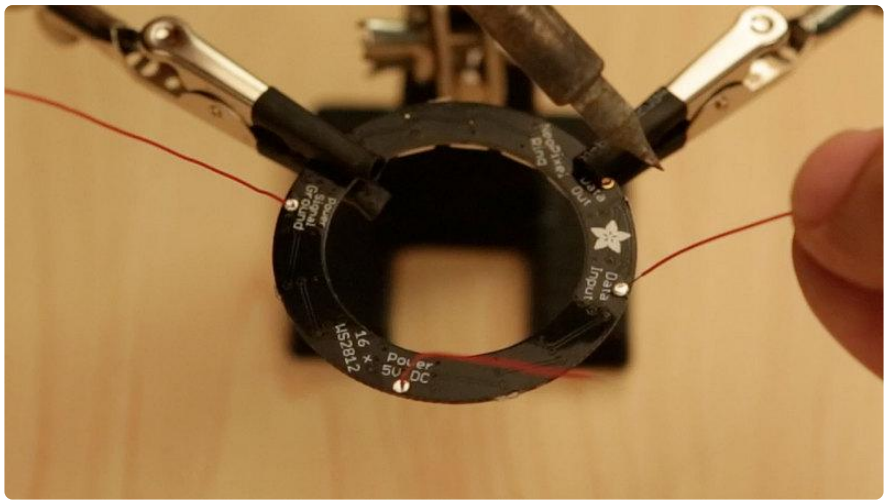
Soldering Circuit

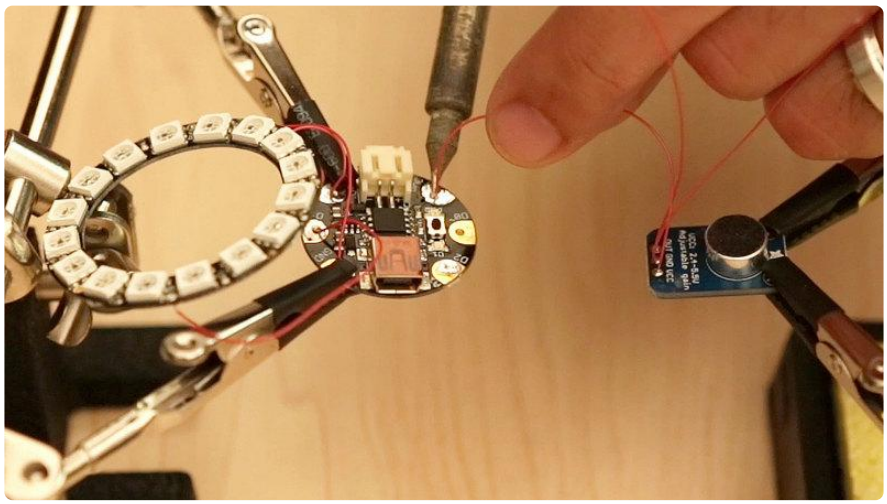
First we need to measure the length of 20 gage wires. These wires are pretty thin. The minimum length of these wires should be around 60mm (2.3in). This should be long enough to position each component separately from each other but short enough not to clutter the inside of the frame. Cut and strip 6 pieces of wire (3 for the mic sensor and 3 for the NeoPixel ring). We've found the soldering process is a bit easier if you solder the wires to the NeoPixel Ring and mic sensor first, and then those wires to the Gemma. You can tin your connections by adding small amounts of solder to the pins. You can use a handy tool to hold your components while you solder.

Since the NeoPixel Ring is mounted to the bottom cover, you'd get better lighting quality if you solder the wires coming from the back of the NeoPixel ring PCB. The mic sensor can have the wires soldered to the front of the PCB.

Once the mic sensor and NeoPixel ring have the wires securely soldered, use the handy tool to hold the components and solder them to the Gemma. The mic sensor and neopixel ring are going to share the ground **GND** pin on the Gemma.

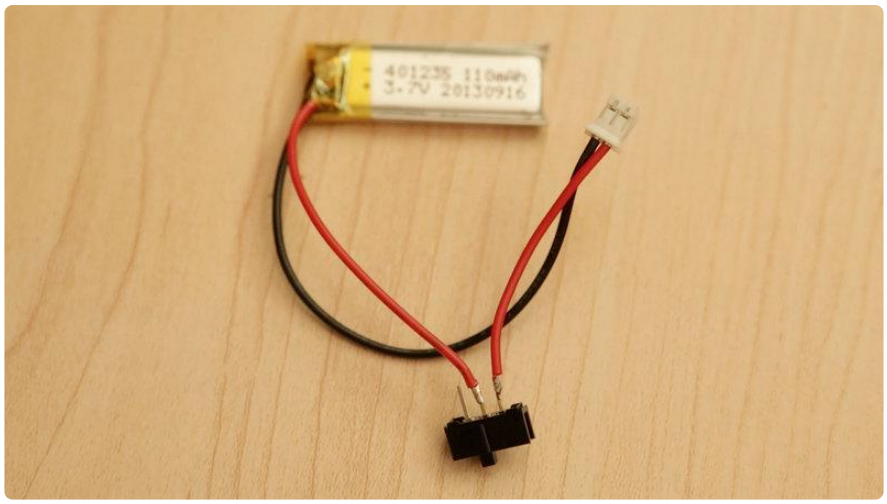




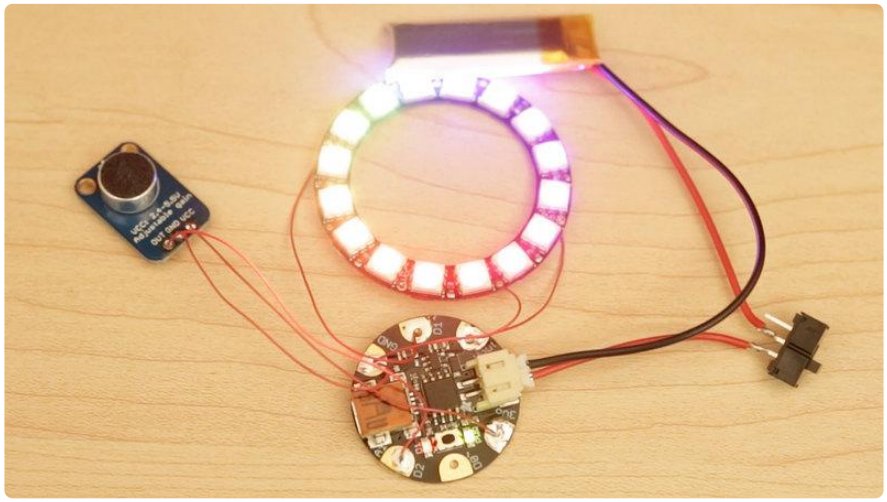


Battery + Switch

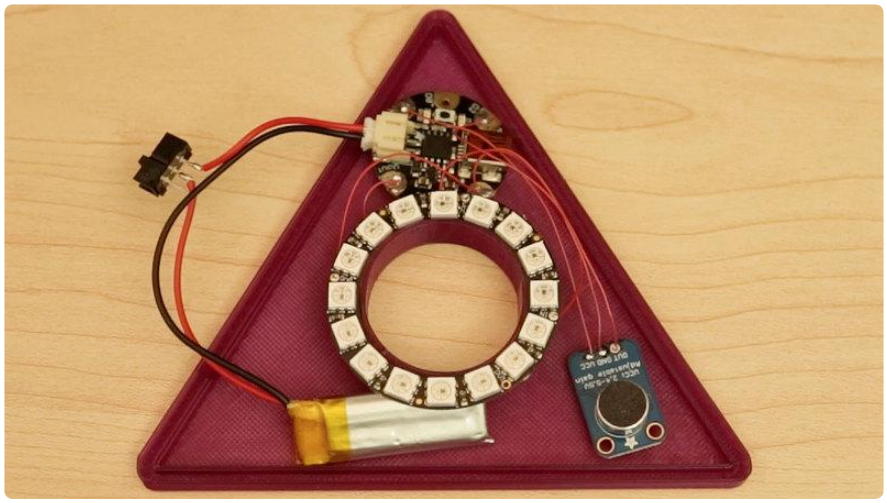
The slide switch needs to be connected to the lithium polymer battery so that the circuit can easily be powered on and off. To do this, first cut the RED wire connect of the lithium polymer battery. Now you will need to solder and connect one end of the red wire to the middle and left (or right) pin of the slide switch. See our circuit diagram to get a better visual idea. Please be very careful when you do this! Try hard not to pull out the wires from the battery and never let the red and black wires touch!



With the power circuit assembled, plug in the lithium recharage battery to the JST port of the Gemma. Slide the switch and it should power on!



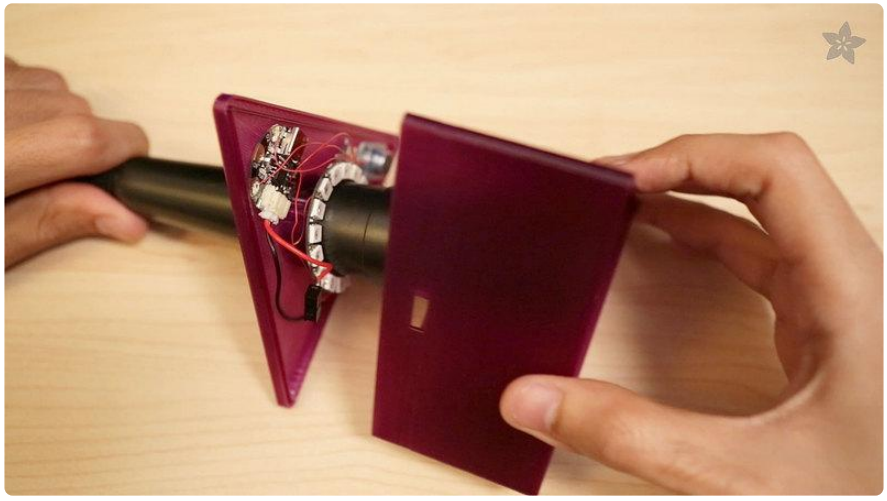
Assembly



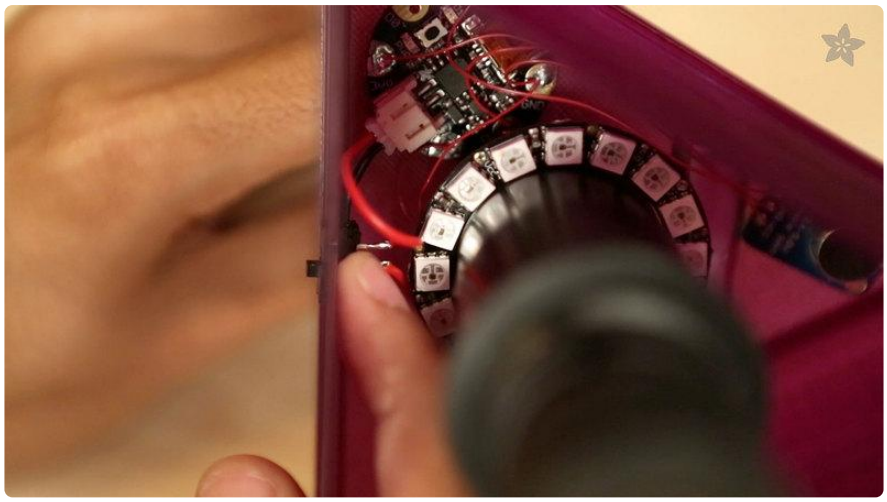
Position the working circuit onto the bottom cover. It's the one with the smaller hole cut out and the extruded pipe we made in 123D Design. Since the design is

symmetrical, the covers will snap right onto the frame. The parts should be positioned close to the ends like shown in the photo above. The NeoPixel Ring should snuggly rest on top of the extruded pipe. Add pieces of foam tape to the bottom of the components to secure them in place. Don't ever let the components or wires touch!

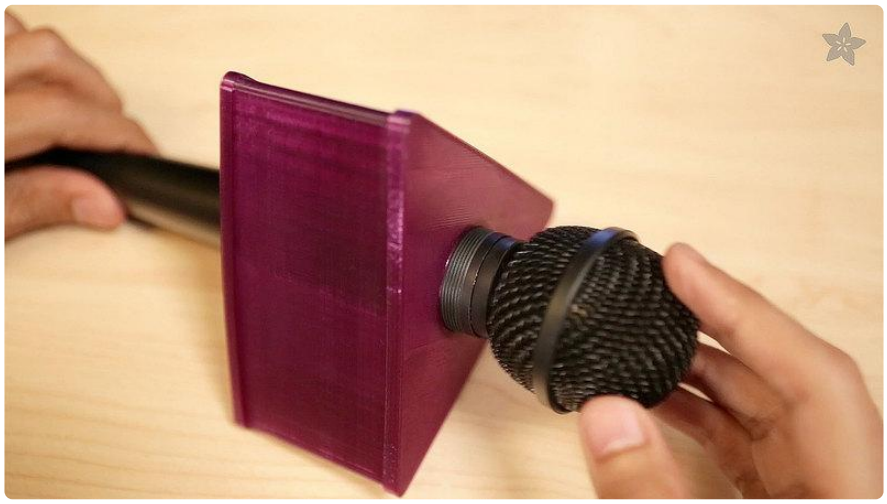
Slide the hand held microphone from the bottom into the bottom cover. If you have a wired mic, string it down through the hole. The tolerance should be tight enough to hold the bottom flag piece in place. Take caution not to snag or damage any of the wiring from the circuit, be careful doing this!



Position the frame over the mic and align the slide switch up to the opening on the frame. The frame should tightly snap onto the bottom cover, securing it into place. If your microphone has a removable pop filter, now is a good time to remove it (most of them unscrew).



Use your index finger or tweezers to lightly push the slide switch through the cutout of the frame. The switch should have a nice fit but you'll want to use an adhesive to secure the slide switch into the frame. Be very careful not to let the pins of the switch touch the NeoPixel ring PCB!



Position the top cover over the mic and push it through the top of the mic. Gently snap the top cover piece into the frame. Now you can twist on the microphone filter until it securely holds the top piece in place.

To personalize your 3D Printed mic flag, you can add a logo or text to one or all sides of the frame. The name or logo can be 3D printed or crafted with fabric or other neat materials! Remember, you can achieve a better sound reactive response by adjusting the mic amp gain until it only picks up audio that is the closest to it.

Arduino Code

The Arduino code presented below works equally well on all versions of GEMMA: v2 and M0. But if you have an M0 board, consider using the CircuitPython code on the next page of this guide, no Arduino IDE required! Click to Download the NeoPixel Library

Installing Arduino libraries is a frequent stumbling block. If this is your first time, or simply needing a refresher, please read the All About Arduino Libraries (https://adafru.it/aYM) tutorial. (https://adafru.it/zAX)If the library is correctly installed (and the Arduino IDE is restarted), you should be able to navigate through the "File" rollover menus as follows:

File→Sketchbook→Libraries→Adafruit_NeoPixel→strandtest

```
// SPDX-FileCopyrightText: 2013 Phil Burgess for Adafruit Industries
//
// SPDX-License-Identifier: BSD-3-Clause
/*
LED VU meter for Arduino and Adafruit NeoPixel LEDs.

 Hardware requirements:
 - Most Arduino or Arduino-compatible boards (ATmega 328P or better).
 - Adafruit Electret Microphone Amplifier (ID: 1063)
 - Adafruit Flora RGB Smart Pixels (ID: 1260)
 OR
 - Adafruit NeoPixel Digital LED strip (ID: 1138)
 - Optional: battery for portable use (else power through USB or adapter)
 Software requirements:
 - Adafruit NeoPixel library
```

```
 Connections:
 - 3.3V to mic amp +
 - GND to mic amp -
 - Analog pin to microphone output (configurable below)
 - Digital pin to LED data input (configurable below)
 See notes in setup() regarding 5V vs. 3.3V boards - there may be an
 extra connection to make and one line of code to enable or disable.

 Written by Adafruit Industries.  Distributed under the BSD license.
 This paragraph must be included in any redistribution.

 fscale function:
 Floating Point Autoscale Function V0.1
 Written by Paul Badger 2007
 Modified from code by Greg Shakar

 */

#include <Adafruit_NeoPixel.h>
#include <math.h>

#define N_PIXELS  16  // Number of pixels in strand
#define MIC_PIN   A1  // Microphone is attached to this analog pin
#define LED_PIN    1  // NeoPixel LED strand is connected to this pin
#define SAMPLE_WINDOW   10  // Sample window for average level
#define PEAK_HANG 24 //Time of pause before peak dot falls
#define PEAK_FALL 4 //Rate of falling peak dot
#define INPUT_FLOOR 10 //Lower range of analogRead input
#define INPUT_CEILING 300 //Max range of analogRead input, the lower the value the
more sensitive (1023 = max)



byte peak = 16;      // Peak level of column; used for falling dots
unsigned int sample;

byte dotCount = 0;  //Frame counter for peak dot
byte dotHangCount = 0; //Frame counter for holding peak dot

Adafruit_NeoPixel strip = Adafruit_NeoPixel(N_PIXELS, LED_PIN, NEO_GRB +
NEO_KHZ800);

void setup()
{
  // This is only needed on 5V Arduinos (Uno, Leonardo, etc.).
  // Connect 3.3V to mic AND TO AREF ON ARDUINO and enable this
  // line.  Audio samples are 'cleaner' at 3.3V.
  // COMMENT OUT THIS LINE FOR 3.3V ARDUINOS (FLORA, ETC.):
  //  analogReference(EXTERNAL);

  // Serial.begin(9600);
  strip.begin();
  strip.show(); // Initialize all pixels to 'off'

}

void loop()
{
  unsigned long startMillis= millis();  // Start of sample window
  float peakToPeak = 0;   // peak-to-peak level

  unsigned int signalMax = 0;
  unsigned int signalMin = 1023;
  unsigned int c, y;


  // collect data for length of sample window (in mS)
  while (millis() - startMillis < SAMPLE_WINDOW)
  {
```

```
    sample = analogRead(MIC_PIN);
    if (sample < 1024)  // toss out spurious readings
    {
      if (sample > signalMax)
      {
        signalMax = sample;  // save just the max levels
      }
      else if (sample < signalMin)
      {
        signalMin = sample;  // save just the min levels
      }
    }
  }
  peakToPeak = signalMax - signalMin;  // max - min = peak-peak amplitude

  // Serial.println(peakToPeak);


  //Fill the strip with rainbow gradient
  for (int i=0;i<=strip.numPixels()-1;i++){
    strip.setPixelColor(i,Wheel(map(i,0,strip.numPixels()-1,30,150)));
  }


  //Scale the input logarithmically instead of linearly
  c = fscale(INPUT_FLOOR, INPUT_CEILING, strip.numPixels(), 0, peakToPeak, 2);




  if(c < peak) {
    peak = c;        // Keep dot on top
    dotHangCount = 0;    // make the dot hang before falling
  }
  if (c <= strip.numPixels()) { // Fill partial column with off pixels
    drawLine(strip.numPixels(), strip.numPixels()-c, strip.Color(0, 0, 0));
  }

  // Set the peak dot to match the rainbow gradient
  y = strip.numPixels() - peak;

  strip.setPixelColor(y-1,Wheel(map(y,0,strip.numPixels()-1,30,150)));

  strip.show();

  // Frame based peak dot animation
  if(dotHangCount > PEAK_HANG) { //Peak pause length
    if(++dotCount >= PEAK_FALL) { //Fall rate
      peak++;
      dotCount = 0;
    }
  }
  else {
    dotHangCount++;
  }
}

//Used to draw a line between two points of a given color
void drawLine(uint8_t from, uint8_t to, uint32_t c) {
  uint8_t fromTemp;
  if (from > to) {
    fromTemp = from;
    from = to;
    to = fromTemp;
  }
  for(int i=from; i<=to; i++){
    strip.setPixelColor(i, c);
  }
}
```

```
float fscale( float originalMin, float originalMax, float newBegin, float
newEnd, float inputValue, float curve){

  float OriginalRange = 0;
  float NewRange = 0;
  float zeroRefCurVal = 0;
  float normalizedCurVal = 0;
  float rangedValue = 0;
  boolean invFlag = 0;


  // condition curve parameter
  // limit range

  if (curve > 10) curve = 10;
  if (curve < -10) curve = -10;

  curve = (curve * -.1) ; // - invert and scale - this seems more intuitive -
postive numbers give more weight to high end on output
  curve = pow(10, curve); // convert linear scale into lograthimic exponent for
other pow function

  /*
   Serial.println(curve * 100, DEC);   // multply by 100 to preserve resolution
   Serial.println();
   */

  // Check for out of range inputValues
  if (inputValue < originalMin) {
    inputValue = originalMin;
  }
  if (inputValue > originalMax) {
    inputValue = originalMax;
  }

  // Zero Refference the values
  OriginalRange = originalMax - originalMin;

  if (newEnd > newBegin){
    NewRange = newEnd - newBegin;
  }
  else
  {
    NewRange = newBegin - newEnd;
    invFlag = 1;
  }

  zeroRefCurVal = inputValue - originalMin;
  normalizedCurVal  =  zeroRefCurVal / OriginalRange;   // normalize to 0 - 1 float

  // Check for originalMin > originalMax  - the math for all other cases i.e.
negative numbers seems to work out fine
  if (originalMin > originalMax ) {
    return 0;
  }

  if (invFlag == 0){
    rangedValue =  (pow(normalizedCurVal, curve) * NewRange) + newBegin;

  }
  else     // invert the ranges
  {
    rangedValue =  newBegin - (pow(normalizedCurVal, curve) * NewRange);
  }

  return rangedValue;
}
```

```
// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  if(WheelPos < 85) {
    return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  }
  else if(WheelPos < 170) {
    WheelPos -= 85;
    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  else {
    WheelPos -= 170;
    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
}
```

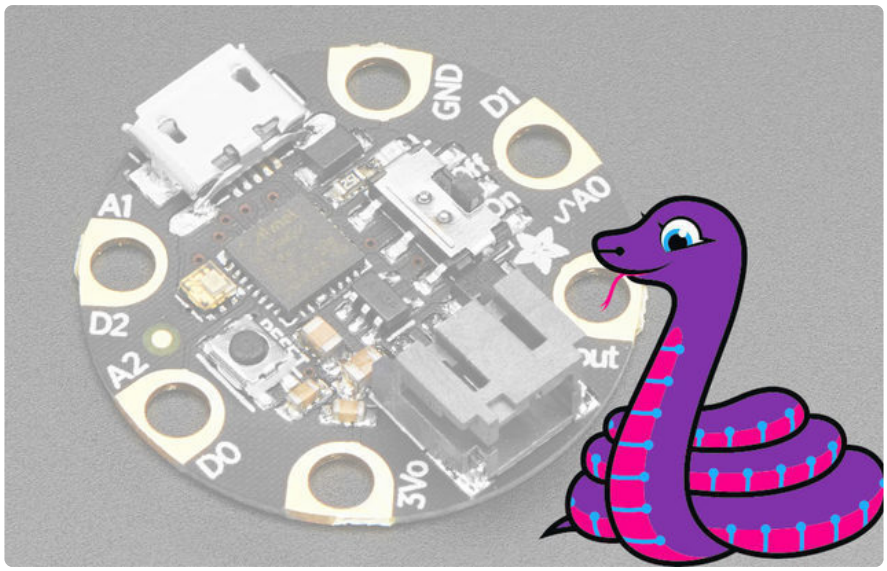
From the **Tools→Board** menu, select the device you are using:

- **Adafruit Gemma M0**
- **Adafruit Gemma 8 MHz**

Connect the USB cable between the computer and your device. The original Gemma (8 MHz) need the reset button pressed on the board, then click the upload button (right arrow icon) in the Arduino IDE. You do not need to press the reset on the newer Gemma M0.

When the battery is connected, you should get a light show from the LEDs. All your pixels working? Great! You can take apart this prototype and get ready to put the pixels in the collar. Refer to the NeoPixel Uberguide (https://adafru.it/dhw) for more info.

CircuitPython Code



GEMMA M0 boards can run **CircuitPython** — a different approach to programming compared to Arduino sketches. In fact, **CircuitPython comes factory pre-loaded on GEMMA M0**. If you've overwritten it with an Arduino sketch, or just want to learn the basics of setting up and using CircuitPython, this is explained in the **Adafruit GEMMA M0 guide** (https://adafru.it/z1B).

These directions are specific to the "M0" GEMMA board. The original GEMMA with an 8-bit AVR microcontroller doesn't run CircuitPython...for those boards, use the Arduino sketch on the "Arduino code" page of this guide.

Below is CircuitPython code that works similarly (though not exactly the same) as the Arduino sketch shown on a prior page. To use this, plug the GEMMA M0 into USB...it should show up on your computer as a small **flash drive**...then edit the file "**code.py**" with your text editor of choice. Select and copy the code below and paste it into that file, **entirely replacing its contents** (don't mix it in with lingering bits of old code). When you save the file, the code should **start running almost immediately** (if not, see notes at the bottom of this page).

If GEMMA M0 doesn't show up as a drive, follow the GEMMA M0 guide link above to prepare the board for CircuitPython.

```
# SPDX-FileCopyrightText: 2013 Phil Burgess for Adafruit Industries
# SPDX-FileCopyrightText: 2017 Mikey Sklar for Adafruit Industries
#
# SPDX-License-Identifier: BSD-3-Clause

# LED VU meter for Arduino and Adafruit NeoPixel LEDs.

# Hardware requirements:
```

```
# - M0 boards
# - Adafruit Electret Microphone Amplifier (ID: 1063)
# - Adafruit Flora RGB Smart Pixels (ID: 1260)
# OR
# - Adafruit NeoPixel Digital LED strip (ID: 1138)
# - Optional: battery for portable use (else power through USB or adapter)
# Software requirements:
# - Adafruit NeoPixel library

# Connections:
# - 3.3V to mic amp +
# - GND to mic amp -
# - Analog pin to microphone output (configurable below)
# - Digital pin to LED data input (configurable below)
# See notes in setup() regarding 5V vs. 3.3V boards - there may be an
# extra connection to make and one line of code to enable or disable.

# Written by Adafruit Industries.  Distributed under the BSD license.
# This paragraph must be included in any redistribution.

# fscale function:
# Floating Point Autoscale Function V0.1
# Written by Paul Badger 2007
# Modified fromhere code by Greg Shakar
# Ported to Circuit Python by Mikey Sklar

import time

import board
import neopixel
from rainbowio import colorwheel
from analogio import AnalogIn

n_pixels = 16  # Number of pixels you are using
mic_pin = AnalogIn(board.A1)  # Microphone is attached to this analog pin
led_pin = board.D1  # NeoPixel LED strand is connected to this pin
sample_window = .1  # Sample window for average level
peak_hang = 24  # Time of pause before peak dot falls
peak_fall = 4  # Rate of falling peak dot
input_floor = 10  # Lower range of analogRead input
# Max range of analogRead input, the lower the value the more sensitive
# (1023 = max)
input_ceiling = 300

peak = 16  # Peak level of column; used for falling dots
sample = 0

dotcount = 0  # Frame counter for peak dot
dothangcount = 0  # Frame counter for holding peak dot

strip = neopixel.NeoPixel(led_pin, n_pixels, brightness=1, auto_write=False)


def remapRange(value, leftMin, leftMax, rightMin, rightMax):
    # this remaps a value fromhere original (left) range to new (right) range
    # Figure out how 'wide' each range is
    leftSpan = leftMax - leftMin
    rightSpan = rightMax - rightMin

    # Convert the left range into a 0-1 range (int)
    valueScaled = int(value - leftMin) / int(leftSpan)

    # Convert the 0-1 range into a value in the right range.
    return int(rightMin + (valueScaled * rightSpan))


def fscale(originalmin, originalmax, newbegin, newend, inputvalue, curve):
    invflag = 0
```

```
    # condition curve parameter
    # limit range
    if curve > 10:
        curve = 10
    if curve < -10:
        curve = -10

    # - invert and scale -
    # this seems more intuitive
    # postive numbers give more weight to high end on output
    curve = (curve * -.1)
    # convert linear scale into lograthimic exponent for other pow function
    curve = pow(10, curve)

    # Check for out of range inputValues
    if inputvalue < originalmin:
        inputvalue = originalmin

    if inputvalue > originalmax:
        inputvalue = originalmax

    # Zero Refference the values
    originalrange = originalmax - originalmin

    if newend > newbegin:
        newrange = newend - newbegin
    else:
        newrange = newbegin - newend
        invflag = 1

    zerorefcurval = inputvalue - originalmin
    # normalize to 0 - 1 float
    normalizedcurval = zerorefcurval / originalrange

    # Check for originalMin > originalMax
    # -the math for all other cases
    # i.e. negative numbers seems to work out fine
    if originalmin > originalmax:
        return 0

    if invflag == 0:
        rangedvalue = (pow(normalizedcurval, curve) * newrange) + newbegin
    else:  # invert the ranges
        rangedvalue = newbegin - (pow(normalizedcurval, curve) * newrange)

    return rangedvalue


def drawLine(fromhere, to):
    if fromhere > to:
        to, fromhere = fromhere, to

    for index in range(fromhere, to):
        strip[index] = (0, 0, 0)


while True:

    time_start = time.monotonic()  # current time used for sample window
    peaktopeak = 0  # peak-to-peak level
    signalmax = 0
    signalmin = 1023
    c = 0
    y = 0

    # collect data for length of sample window (in seconds)
    while (time.monotonic() - time_start) < sample_window:

        # convert to arduino 10-bit [1024] fromhere 16-bit [65536]
```

```
        sample = mic_pin.value / 64

        if sample < 1024:  # toss out spurious readings

            if sample > signalmax:
                signalmax = sample  # save just the max levels
            elif sample < signalmin:
                signalmin = sample  # save just the min levels

    peaktopeak = signalmax - signalmin  # max - min = peak-peak amplitude

    # Fill the strip with rainbow gradient
    for i in range(0, len(strip)):
        strip[i] = colorwheel(remapRange(i, 0, (n_pixels - 1), 30, 150))

    # Scale the input logarithmically instead of linearly
    c = fscale(input_floor, input_ceiling, (n_pixels - 1), 0, peaktopeak, 2)

    if c < peak:
        peak = c  # keep dot on top
        dothangcount = 0  # make the dot hang before falling

    if c <= n_pixels:  # fill partial column with off pixels
        drawLine(n_pixels, n_pixels - int(c))

    # Set the peak dot to match the rainbow gradient
    y = n_pixels - peak
    strip.fill = (y - 1, colorwheel(remapRange(y, 0, (n_pixels - 1), 30, 150)))
    strip.write()

    # Frame based peak dot animation
    if dothangcount > peak_hang:  # Peak pause length
        dotcount += 1
        if dotcount >= peak_fall:  # Fall rate
            peak += 1
            dotcount = 0
    else:
        dothangcount += 1
```

This code requires the **neopixel.py** library. A factory-fresh board will have this already installed. If you've just reloaded the board with CircuitPython, create the "lib" directory and then download neopixel.py from Github (https://adafru.it/yew).