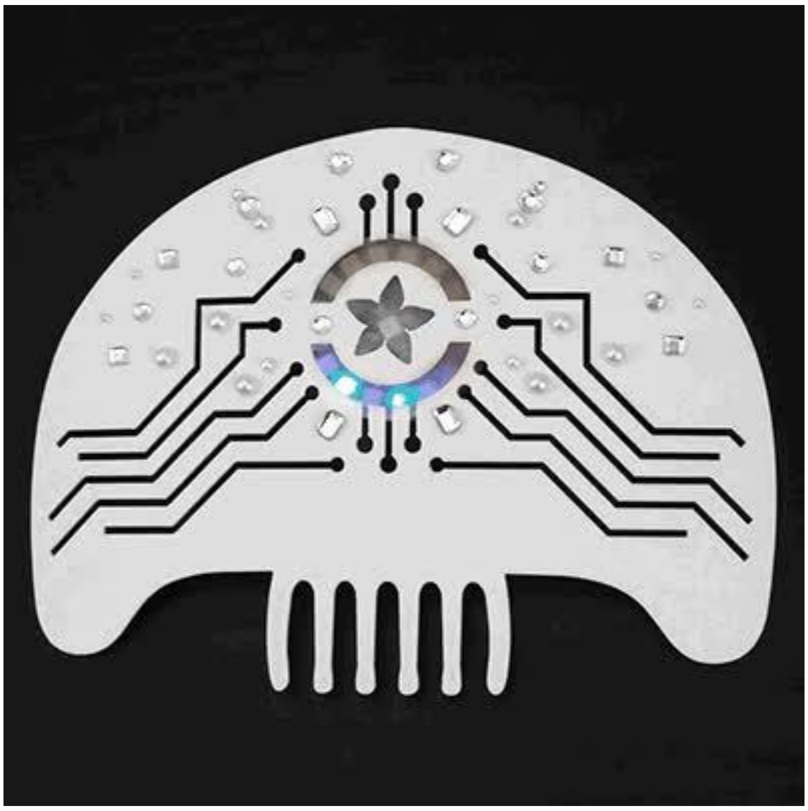
3D Printed NeoPixel Ring Hair Dress

Created by Ruiz Brothers



https://learn.adafruit.com/neopixel-ring-hair-dress

Last updated on 2022-12-01 02:04:38 PM EST

Table of Contents

Overview 3

Customize 4

- Design & Illustration
- Tinkercad

3D Printing 6

- Printing Techniques
- Build Plate Preparations
- Live Level
- Gemma Enclosure
- Build Plate Preparations

Bedazzle 8

Arduino Code 9

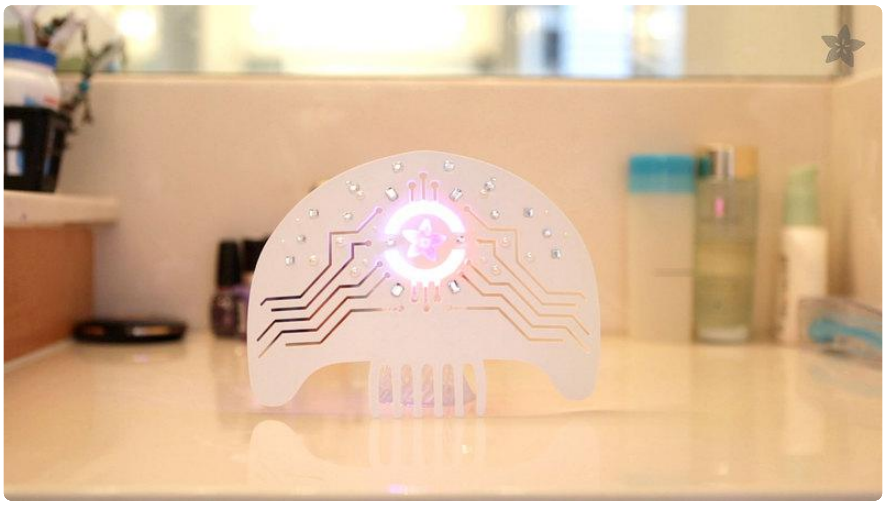
- Program Gemma
- Prototype

CircuitPython Code 10

Circuit Diagram 16

Assemble Hair Dress 18

Overview



Here's an elegant wearables project for those really special occasions. A 3D Printed, LED Hair Dress, powered by an NeoPixel ring and Gemma, Adafruits tiny yet powerful wearables micro-controller.

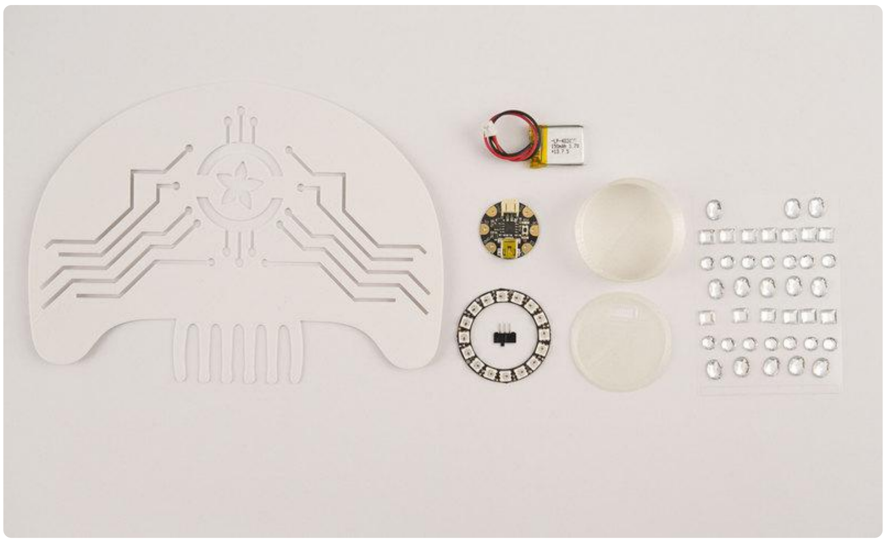
This guide was written for the 'original' Gemma board, but can be done with either the original or M0 Gemma. We recommend the Gemma M0 as it is easier to use and is more compatible with modern computers

Prerequisite guides:

- Introducing GEMMA () or Introducing Gemma M0 guide ()
- NeoPixel Überguide ()

You will need:

- GEMMA board (http://adafru.it/1222) or Trinket (http://adafru.it/1501)
- 16x NeoPixel Ring (http://adafru.it/1463)
- Slide Switch (http://adafru.it/805)
- Thin Wire (http://adafru.it/1446)
- Soldering iron (http://adafru.it/180) and solder (http://adafru.it/145)
- Small LiPo Battery (http://adafru.it/1317)
- MakerBot Adafruit Edition (http://adafru.it/1292)
- Bedazzle Stones ()



Customize

Design & Illustration

You can use our hair dress template or create your own in photoshop, illustrator or any other vector drawling application that can export SVG files. Our hair dress template is 144mm x 195mm. The bottom comb is large enough to rest inside of a hair bun with a little support from a hair clip. The center of the artwork was designed so that a NeoPixel ring can shine through the hair dress. Your hair dress can be designed to fit individual NeoPixels, or even NeoPixel Strips (http://adafru.it/1138)+Sticks (http://adafru.it/1426)!

The illustration inside of the hair dress should be manifold and not have any stray pieces. If your artwork is complex with multiple objects, you should merge them together to create one unified vector path. Your artwork will need to be exported in the SVG file format.

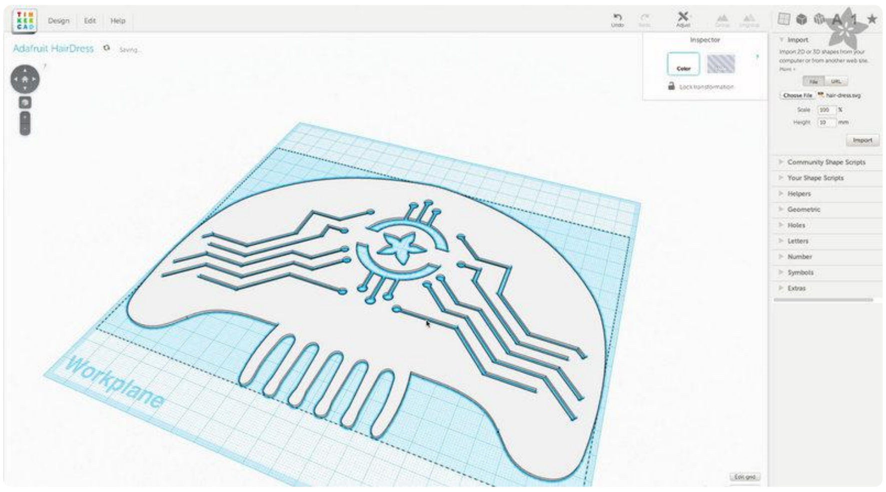
Stuck on the creative stuff? You can get some great inspiration and actual vector files on sites like http://vecteezy.com () and http://all-free-download.com (). The design should complement your event or character for a great cosplay!

Tinkercad

We used tinkercad to turn the SVG into a 3D model that can be printed because its super easy to import vector art and extrude it in 3D. You can import SVG files in tinker by selecting the import drop down menu in the right side and choosing your file by either upload or URL. Drag and use the handles along side the object to resize the hair dress. Use the middle handle point to resize the thickness of the object. The height of the haird ress should be extruded to a minimum of 1mm thick.

When your design is finalized, you will want to flip the design up-side-down so that the artwork is printed facing the heated build plate. The side that gets printed on the heated surface creates an almost seamless texture that makes for a great quality print.

For extra awesome, import separate SVG artwork to create depth and accessories to the hair dress for an epic design! To get familiar with tinkercad's UI, check out their tutorial videos on their youtube channel. It's an easy and powerful CAD web app!



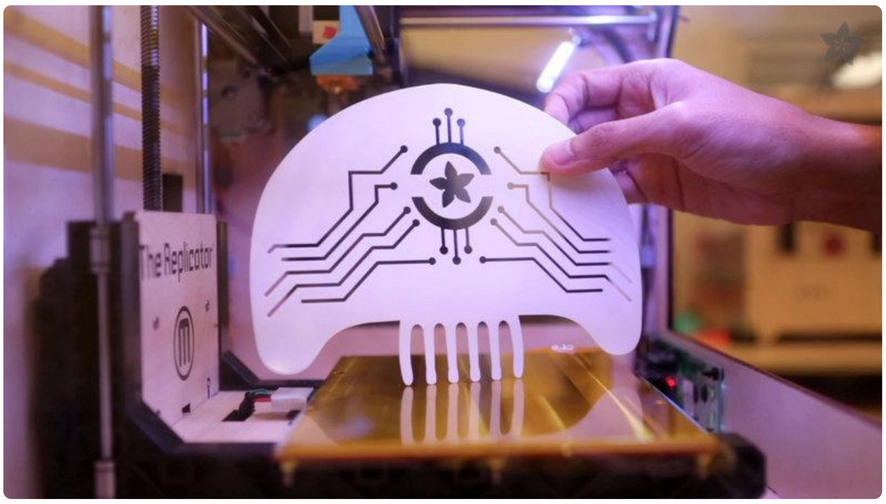
3D Printing

The hair dress is printed in single color, white ABS plastic. But you are free to experiment with multiple colors. A great way to do that is to switch out your filament while printing objects that are higher than the hair dress base.

Ensure your artwork is flipped up-side-down so that it's printed right-side-up when it's removed from the heated build plate. If you forgot to flip it in your 3D app, you can rotate the STL in your slicing program. In MakerWare, you can select the object and click on the turn menu item. Press the +90 button until the hair dress is flipped.

TinkerCad Template

Hair Dress About 1 hr 15 minutes 21g	ABS @230/120 No Raft No Support	20% infill / 2 shells .20 Layer Height 80/120 mm/s



Download STLs

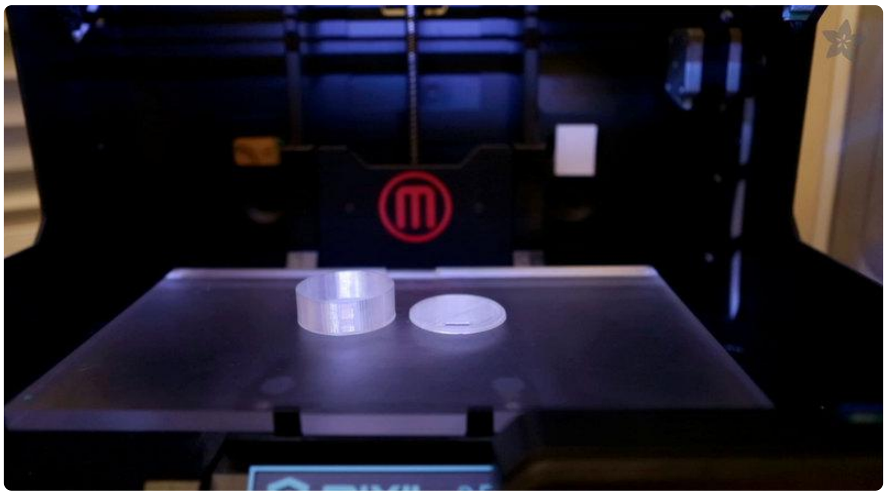
Printing Techniques

Build Plate Preparations

For ABS printing, it's recommenced to use a heated build plate with katon tape. For best quality, apply a fresh strip of katon tape. You can apply a clear-air-bubble sheet by laying the tape down when the build plate is heated (Be careful!). Use a plastic card to flatten the tape. When air-bubble arise, peel back the tape and reapply the tape in small chucks until it's clean and air bubble free.

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.



Gemma Enclosure

To defuse and soften the NeoPixel ring, we printed the Gemma enclosure in transparent PLA filament. Each piece can be printed separately or together. The lid cover should easily snap onto the case and tightly fit so the components don't fall out of the enclosure.

Build Plate Preparations

There's a great video tutorial () 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.

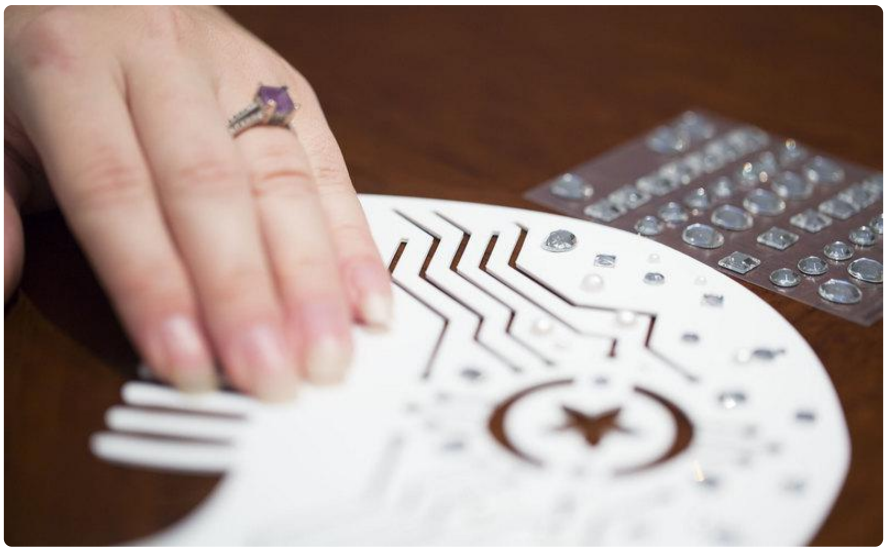
Removing the pieces

The lid will be the most difficult to remove, use a circuit spatula (). Try to avoid scratching your acrylic build plate. A good way to remove the covers is to position the spatula on the edge of the layer above the build plate. Apply pressure to the spatula, closely grip it upwards and pull up removing the piece from the build plate.

Gemma Enclosure About 30 minutes 4g	PLA @230 No Raft No Support	.20 Layer Hieght 90/150mm/s

Bedazzle

We picked up a pack of fake diamonds and perls () from our local crafty arts supply store for under $3 bucks, but you are more than welcome to use any glitter, stickers, stones what-have-you on your hair dress. If you want to paint the hair dress, be sure to use acrylic based paints thats sure to stick.



We placed diamonds along side the edges and in between the spots of the circuit cut outs. The pearls were sparingly applied to each side to fill in the gaps of the design. Your design can have as many stones as you like, just remember to super glue them on!

To create a slight curvature to the hair dress, we gently bent the plastic. Since it's 1mm thick and printed in ABS, it's flexible and durable enough not to snap. We used our handy vise gripper and held it bent over night. This creates a slight curve that makes the hair dress more comfortable when wearing. We found this to be a better alternative to using a heat gun because the ABS material tends to curl and cause warping.

Arduino Code

The Arduino code presented below works equally well on all versions of GEMMA: v1, 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!

Program Gemma

We used the code from GitHub user HerrRausB () to generate complex animations in the NeoPixel ring. You can download his code from GitHub. The code is nicely documented and has editable parameters that can be customized to fit your project. Huge thanks to HerrRausB!

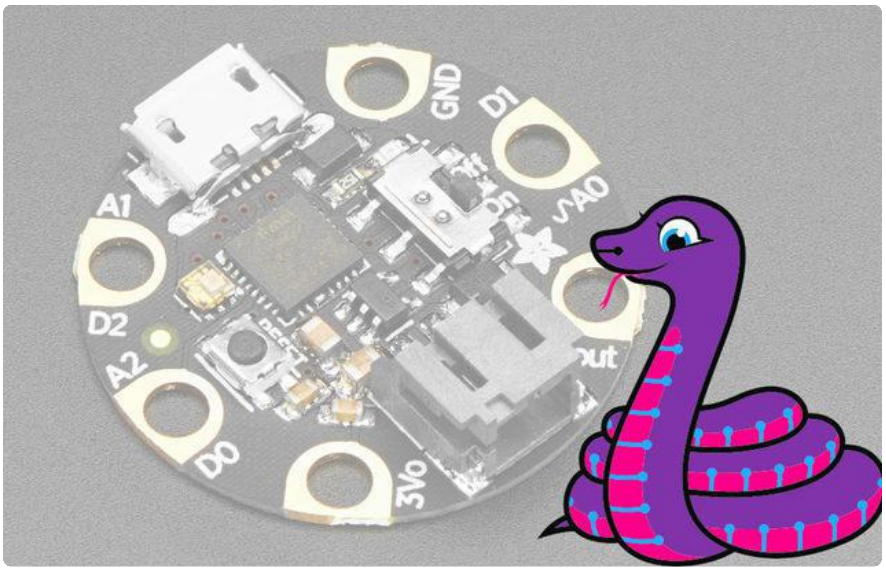
Once you have the sketches how you want, you will need to upload the code to the Gemma. Please follow the instructions on the Gemma introduction () guide to get your Arduino IDE setup with the Gemma. Once you have that sorted out, you will need to push the reset button on the Gemma and then hit the upload sketch button(the arrow icon next to check mark) in the Arduino IDE.

Download Gemma Hoop Animator

Prototype

If your unfamiliar with connecting the Gemma to a NeoPixel ring, you should prototype the circuit by wiring the components together with alligator clips. On the Gemma, wire up the pin labeled "D0" to the "Data Input" pin on the NeoPixel Ring. Next, wire up ground ("GND") pin of the Gemma to the "Power Signal Ground" pin on the NeoPixel Ring. Now you can connect the Power 5V DC pin on the NeoPixel Ring to the "Vout" pin of the Gemma. You can now plug-in a USB cable from your computer to the Gemma to power it on for prototyping.

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 ().

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.

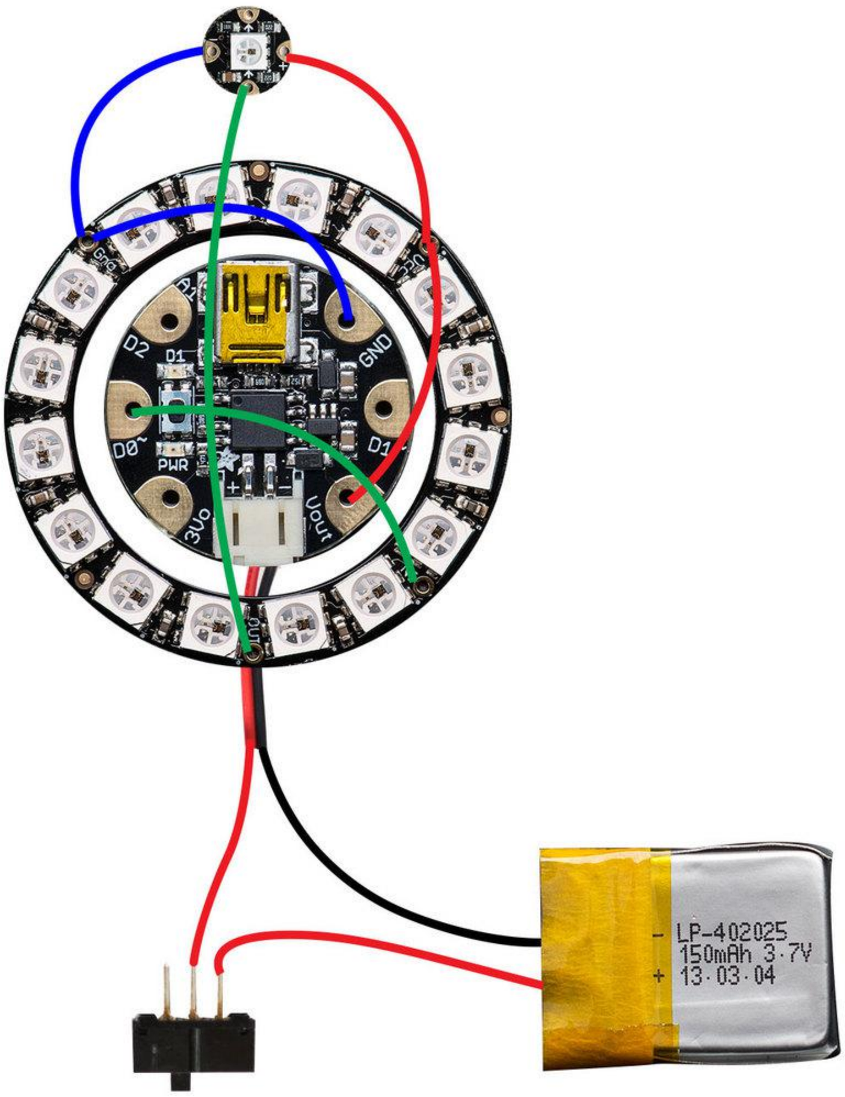
This code requires the neopixel.py library. A factory-fresh board will have this already installed. If you've just reloaded the board with CircuitPython, create the "lib" directory and then download neopixel.py from Github ().

```
# SPDX-FileCopyrightText: 2014 HerrRausB https://github.com/HerrRausB
# SPDX-FileCopyrightText: 2017 Mikey Sklar for Adafruit Industries
#
# SPDX-License-Identifier: LGPL-3.0-or-later
#
# 3D_Printed_NeoPixel_Ring_Hair_Dress.py
#
# this was ported to CircuitPython from the 'Gemma Hoop Animator'
#
# https://github.com/HerrRausB/GemmaHoopAnimator
#
# unless you # don't like the preset animations or find a
# major bug, you don't need tochange anything here
#
import time

import board
import neopixel

try:
    import urandom as random  # for v1.0 API support
except ImportError:
    import random
from analogio import AnalogIn

# available actions
ACT_NOP = 0x00  # all leds off, do nothing
ACT_SIMPLE_RING = 0x01  # all leds on
ACT_CYCLING_RING_ACLK = 0x02  # anti clockwise cycling colors
ACT_CYCLING_RING_CLKW = 0x04  # clockwise cycling colors
ACT_WHEEL_ACLK = 0x08  # anti clockwise spinning wheel
ACT_WHEEL_CLKW = 0x10  # clockwise spinning wheel
ACT_SPARKLING_RING = 0x20  # sparkling effect

numpix = 16  # total number of NeoPixels
pixel_output = board.D0  # pin where NeoPixels are connected
analog_input = board.A0  # needed to seed the random generator
strip = neopixel.NeoPixel(pixel_output, numpix,
                          brightness=.3, auto_write=False)

# available color generation methods
COL_RANDOM = 0x40  # colors will be generated randomly
COL_SPECTRUM = 0x80  # colors will be set as cyclic spectral wipe

# specifiyng the action list
# the action's overall duration in milliseconds (be careful not
action_duration = 0
# to use values > 2^16-1 - roughly one minute :-)

action_and_color_gen = 1  # the color generation method

# the duration of each action step rsp. the delay of the main
action_step_duration = 2
# loop in milliseconds - thus, controls the action speed (be
# careful not to use values > 2^16-1 - roughly one minute :-)

color_granularity = 3  # controls the increment of the R, G, and B
# portions of the rsp. color. 1 means the increment is 0,1,2,3,...,
# 10 means the increment is 0,10,20,... don't use values > 255, and
# note that even values > 127 wouldn't make much sense...

# controls the speed of color changing independently from action
color_interval = 4

# general global variables
color = 0
color_timer = 0
action_timer = 0
```

```
action_step_timer = 0
color_idx = 0
curr_color_interval = 0
curr_action_step_duration = 0
curr_action_duration = 0
curr_action = 0
curr_color_gen = COL_RANDOM
idx = 0
offset = 0
number_of_actions = 31
curr_action_idx = 0
curr_color_granularity = 1
spectrum_part = 0

# defining the animation actions by simply initializing the array of actions
# this array variable must be called theactionlist !!!
#
# valid actions are:
#      ACT_NOP                  simply do nothing and switch everything off
#      ACT_SIMPLE_RING          all leds on
#      ACT_CYCLING_RING_ACLK    anti clockwise cycling colors
#      ACT_CYCLING_RING_CLKW    clockwise cycling colors acording
#      ACT_WHEEL_ACLK           anti clockwise spinning wheel
#      ACT_WHEEL_CLKW           clockwise spinning wheel
#      ACT_SPARKLING_RING       sparkling effect
#
#   valid color options are:
#      COL_RANDOM               colors will be selected randomly, which might
#                               be not very sufficient due to well known
#                               limitations of the random generation algorithm
#      COL_SPECTRUM             colors will be set as cyclic spectral wipe
#                               R -> G -> B -> R -> G -> B -> R -> ...

# action    action name &             action step    color        color change
# duration  color generation method   duration       granularity  interval
theactionlist = [
    [5, ACT_SPARKLING_RING | COL_RANDOM, 0.01, 25, 1],
    [2, ACT_CYCLING_RING_CLKW | COL_RANDOM,
     0.02, 1, 0.005],
    [5, ACT_SPARKLING_RING | COL_RANDOM, 0.01, 25, 1],
    [2, ACT_CYCLING_RING_ACLK | COL_RANDOM,
     0.02, 1, 0.005],
    [5, ACT_SPARKLING_RING | COL_RANDOM, 0.01, 25, 1],
    [2.5, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.25, 20, 0.020],
    [1, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.50, 1, 0.020],
    [.750, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.075, 1, 0.020],
    [.500, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.100, 1, 0.020],
    [.500, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.125, 1, 0.020],
    [.500, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.150, 1, 0.050],
    [.500, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.175, 1, 0.100],
    [.500, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.200, 1, 0.200],
    [.750, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.225, 1, 0.250],
    [1, ACT_CYCLING_RING_CLKW | COL_SPECTRUM,
     0.250, 1, 0.350],
    [30, ACT_SIMPLE_RING | COL_SPECTRUM,
     0.050, 1, 0.010],
    [2.5, ACT_WHEEL_ACLK | COL_SPECTRUM,
     0.010, 1, 0.010],
    [2.5, ACT_WHEEL_ACLK | COL_SPECTRUM,
     0.015, 1, 0.020],
    [2, ACT_WHEEL_ACLK | COL_SPECTRUM,
     0.025, 1, 0.030],
    [1, ACT_WHEEL_ACLK | COL_SPECTRUM,
     0.050, 1, 0.040],
    [1, ACT_WHEEL_ACLK | COL_SPECTRUM,
     0.075, 1, 0.040],
    [1, ACT_WHEEL_ACLK | COL_SPECTRUM,
     0.100, 1, 0.050],
    [.500, ACT_WHEEL_ACLK | COL_SPECTRUM,
     0.125, 1, 0.060],
    [.500, ACT_WHEEL_CLKW | COL_SPECTRUM,
     0.125, 5, 0.050],
    [1, ACT_WHEEL_CLKW | COL_SPECTRUM,
     0.100, 10, 0.040],
    [1.5, ACT_WHEEL_CLKW | COL_SPECTRUM,
     0.075, 15, 0.030],
    [2, ACT_WHEEL_CLKW | COL_SPECTRUM,
     0.050, 20, 0.020],
    [2.5, ACT_WHEEL_CLKW | COL_SPECTRUM,
     0.025, 25, 0.010],
    [3, ACT_WHEEL_CLKW | COL_SPECTRUM,
     0.010, 30, 0.005],
    [5, ACT_SPARKLING_RING | COL_RANDOM, 0.010, 25, 1],
    [5, ACT_NOP, 0, 0, 0]
]

# pylint: disable=global-statement
def nextspectrumcolor():
    global spectrum_part, color_idx, curr_color_granularity, color

    # spectral wipe from green to red
    if spectrum_part == 2:
        color = (color_idx, 0, 255-color_idx)
        color_idx += curr_color_granularity
        if color_idx > 255:
            spectrum_part = 0
            color_idx = 0

    # spectral wipe from blue to green
    elif spectrum_part == 1:
        color = (0, 255 - color_idx, color_idx)
        color_idx += curr_color_granularity
        if color_idx > 255:
            spectrum_part = 2
            color_idx = 0

    # spectral wipe from red to blue
    elif spectrum_part == 0:
        color = (255 - color_idx, color_idx, 0)
        color_idx += curr_color_granularity
        if color_idx > 255:
            spectrum_part = 1
            color_idx = 0


def nextrandomcolor():
    global color

    # granularity = 1 --> [0 .. 255] * 1 --> 0,1,2,3 ... 255
    # granularity = 10 --> [0 .. 25] * 10 --> 0,10,20,30 ... 250
    # granularity = 100 --> [0 .. 2] * 100 --> 0,100, 200 (boaring...)
    random_red = random.randint(0, int(256 / curr_color_granularity))
    random_red *= curr_color_granularity

    random_green = random.randint(0, int(256 / curr_color_granularity))
    random_green *= curr_color_granularity

    random_blue = random.randint(0, int(256 / curr_color_granularity))
    random_blue *= curr_color_granularity
    color = (random_red, random_green, random_blue)


def nextcolor():
    # save some RAM for more animation actions
    if curr_color_gen & COL_RANDOM:
        nextrandomcolor()
    else:
        nextspectrumcolor()


def setup():
    # fingers corssed, the seeding makes sense to really get random colors...
    apin = AnalogIn(analog_input)
    random.seed(apin.value)
    apin.deinit()

    # let's go!
    nextcolor()
    strip.write()


setup()

while True:  # Loop forever...

    # do we need to load the next action?
    if (time.monotonic() - action_timer) > curr_action_duration:
        current_action = theactionlist[curr_action_idx]

        curr_action_duration = current_action[action_duration]
        curr_action = current_action[action_and_color_gen] & 0x3F
        curr_action_step_duration = current_action[action_step_duration]
        curr_color_gen = current_action[action_and_color_gen] & 0xC0
        curr_color_granularity = current_action[color_granularity]
        curr_color_interval = current_action[color_interval]
        curr_action_idx += 1

        # take care to rotate the action list!
        curr_action_idx %= number_of_actions
        action_timer = time.monotonic()

    # do we need to change to the next color?
    if (time.monotonic() - color_timer) > curr_color_interval:
        nextcolor()
        color_timer = time.monotonic()

    # do we need to step up the current action?
    if (time.monotonic() - action_step_timer) > curr_action_step_duration:

        if curr_action:

            is_act_cycling = (ACT_CYCLING_RING_ACLK or ACT_CYCLING_RING_CLKW)

            if curr_action == ACT_NOP:
                # rather trivial even tho this will be repeated as long as the
                # NOP continues - i could have prevented it from repeating
                # unnecessarily, but that would mean more code and less
                # space for more actions within the animation
                for i in range(0, numpix):
                    strip[i] = (0, 0, 0)

            elif curr_action == ACT_SIMPLE_RING:
                # even more trivial - just set the new color, if there is one
                for i in range(0, numpix):
                    strip[i] = color

            elif curr_action == is_act_cycling:
```

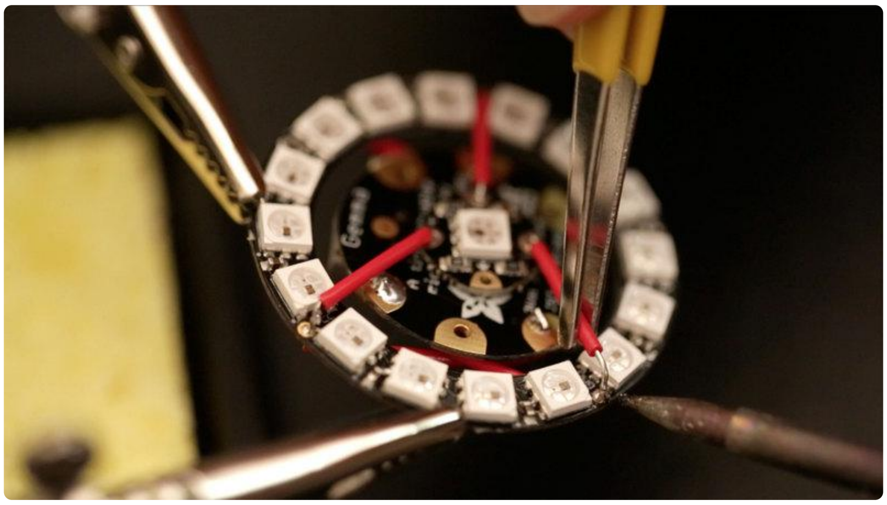
```
            # spin the ring clockwise or anti clockwise
            if curr_action == ACT_CYCLING_RING_ACLK:
                idx += 1
            else:
                idx -= 1

            # prevent overflows or underflows
            idx %= numpix

            # set the new color, if there is one
            strip[idx] = color

        elif curr_action == ACT_WHEEL_ACLK or ACT_WHEEL_CLKW:
            # switch on / off the appropriate pixels according to
            # the current offset
            for idx in range(0, numpix):
                if ((offset + idx) & 7) < 2:
                    strip[idx] = color
                else:
                    strip[idx] = (0, 0, 0)

            # advance the offset and thus, spin the wheel
            if curr_action == ACT_WHEEL_CLKW:
                offset += 1
            else:
                offset -= 1

            # prevent overflows or underflows
            offset %= numpix

        elif curr_action == ACT_SPARKLING_RING:
            # switch current pixel off
            strip[idx] = (0, 0, 0)
            # pick a new pixel
            idx = random.randint(0, numpix)
            # set new pixel to the current color
            strip[idx] = color

    strip.write()
    action_step_timer = time.monotonic()
```

Circuit Diagram

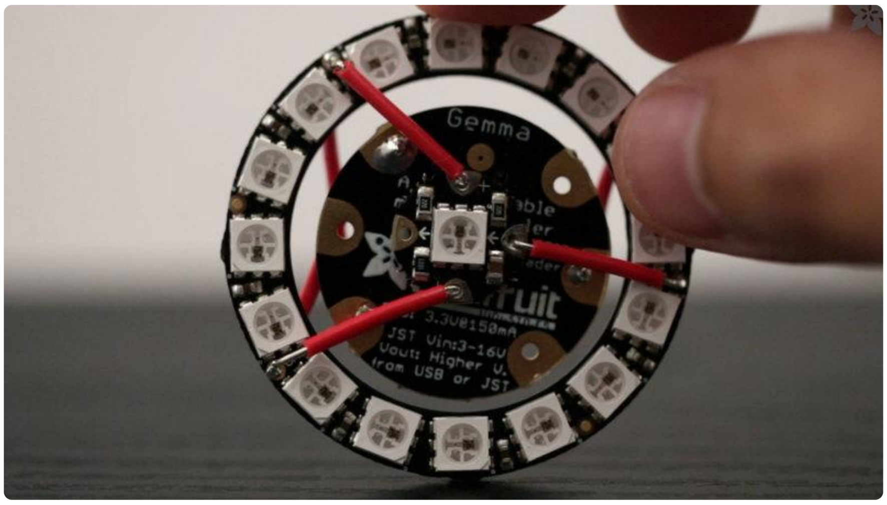


This diagram uses the original Gemma but you can also use the Gemma M0 with the exact same wiring!

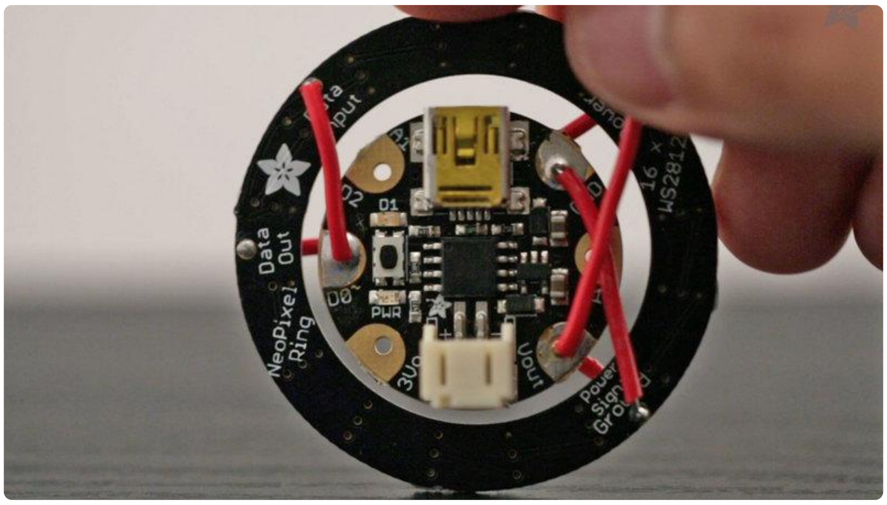
The slide switch needs to be connected to the lithium polymer battery so that the circuit can easily be powered on and off. To do this, first cut the RED wire connect of the lithium polymer battery. Now you will need to solder and connect one end of the red wire to the middle and left (or right) pin 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!



We added one extra NeoPixel in the center of the circuit to complement the design of our hair dress. The pins on the NeoPixel get wired to the back of the Gemma. The Data Out pin on the NeoPixel ring will need to be connected to the Data In pin of the single NeoPixel. The Ground pin of the NeoPixel ring connects to the ground pin of the Gemma, while the Ground pin of the single NeoPixel connects to the ground of the NeoPixel Ring.

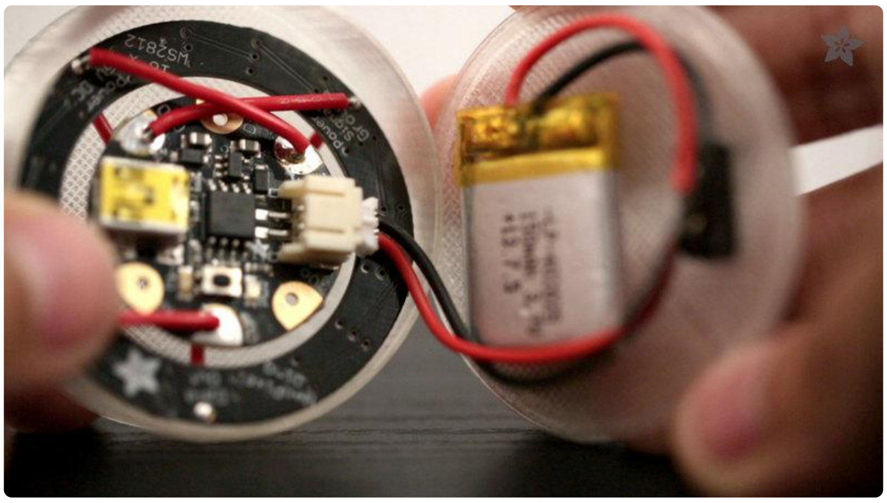


When your circuit is working and tested, you will need to solder the components together. We used 22 gauge wire and measured the lengths needed for each pin and cut them into the appropriate pieces. We used a handy vise tool to keep the Gemma and NeoPixel ring together and tightened the arms so they don't slip while soldering.

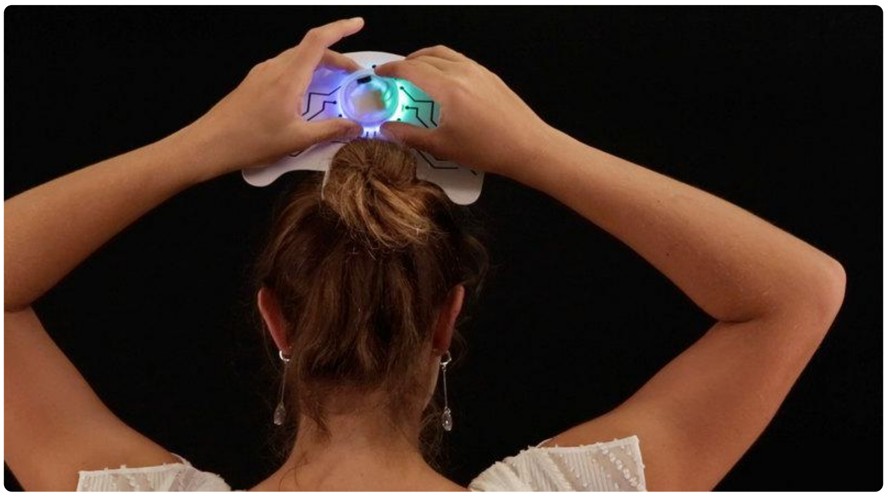


With all the components soldered, the Gemma should have enough support from the wires to hold itself inside of the NeoPixel ring. The Gemma and NeoPixel Ring make a nice independent circuit that can be used for other awesome projects like the Gemma Hoop Earrings ().

Assemble Hair Dress



The Gemma and NeoPixel ring circuit fits nicely inside of the transparent PLA case with the NeoPixels facing down. The NeoPixel ring will become softened and diffused when it shines through the PLA material. Connect the rechargeable lithium polymer battery to the gemma. Now slide the power switch into the case lid so that the switch sticks out. The Power switch will need to be super glued to the lid so that it securely stays in place when it is turned on and off. Allow to the glue to dry for a few minutes. Now you can close the lid, it should securely snap onto the case.



The circuit case can now be attached to the back of the hair dress. We position the case so that the NeoPixel ring shines through the circular ring in the center of the design. We used double-sided sticky foam tape to attach the case to the hair dress so that it can easily be removed for other projects. The foam tape is tuff enough to stick for several hours. You can always use strong adhesives for a more permanent treatment.

To wear the hair dress, simply pull your hair up into a bun and stick it in where it feels most comfortable. You can use hair pins to hold up the hair dress to keep it from flopping over or falling off. Take caution when wearing the hair dress shaking your hear or dancing!