Sound Reactive Sunflower Baby Crib Mobile with Bluetooth Control

Created by Erin St Blaine

https://learn.adafruit.com/sound-reactive-sunflower-baby-crib-mobile-with-bluetooth-control

Last updated on 2024-06-03 03:22:35 PM EDT

Table of Contents

Overview

Make a beautiful hanging mobile with lights for your nursery. This mobile is so cute and so much fun that older kids and adults will definitely enjoy it as well.

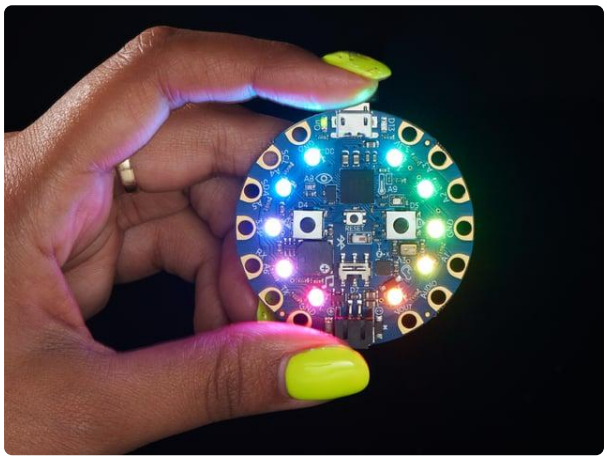
The included sample code has around 8 different animated color modes that you can control via bluetooth using Adafruit's free BlueFruit app. It also has a sound reactive mode, where the lights will pulse in time to music or voices in the room.

This is a fairly easy build that requires just a little bit of tight soldering on the LED strip. The sunflowers are made from craft foam and felt that you can cut by hand or with a vinyl cutting machine. This project uses just a half-meter of very high density NeoPixels, making for gorgeously buttery smooth animations.

Plus, you'll put a Circuit Playground Bluefruit into the hands of the younger generation and start inspiring them to love animated lights from a very young age. That's always a win, in my book.

Parts

This project uses a Circuit Playground Bluefruit microcontroller and 1/2 meter of super high density NeoPixels. The pixel strip has 332 pixels per meter (166 pixels in a half meter), which is enough pixels for five "active" sunflowers.



Circuit Playground Bluefruit - Bluetooth Low Energy

Circuit Playground Bluefruit is our third board in the Circuit Playground series, another step towards a perfect introduction to electronics and programming. We've...

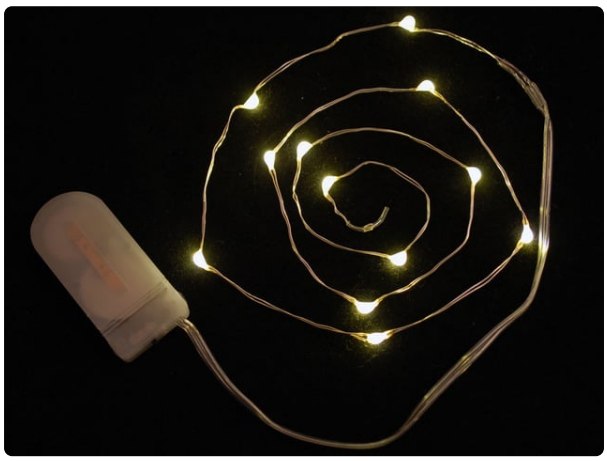
https://www.adafruit.com/product/4333

Adafruit NeoPixel 332 LED-per-Meter Silicone Bead LED Strip

Plug in and glow with this incredible, ultra high density NeoPixel strip with an astonishing 332 LEDs/meter. That's no typo, this half meter strip has 165 miniature NeoPixels...

https://www.adafruit.com/product/4865

I also added two strands of warm white fairy lights, to add visual interest. These can be turned on and off in the code in case you want to add variety to your modes.



Wire Light LED Strand - 10 Warm White LEDs + Coin Cell Holder

Add sparkle to your project with these lovely silver wire LED strands. These strands are very interesting, they use two silver wires that are coated so they don't short if they...

https://www.adafruit.com/product/893

To keep the wiring neat and tidy, I used 30awg wire ribbon cable to run wires from the flowers to the Circuit Playground.

3 x Wire Ribbon Cable — https://www.adafruit.com/product/3889

Silicone Cover Stranded-Core Ribbon Cable - 4 Wires 1 Meter Long - 30 AWG Black

---

You can power with a battery if you want the mobile to spin, or with a USB cable and power plug if you want it to last all year.

1 x Power Supply — https://www.adafruit.com/product/1994

5V 2A Switching Power Supply w/ USB-A Connector

---

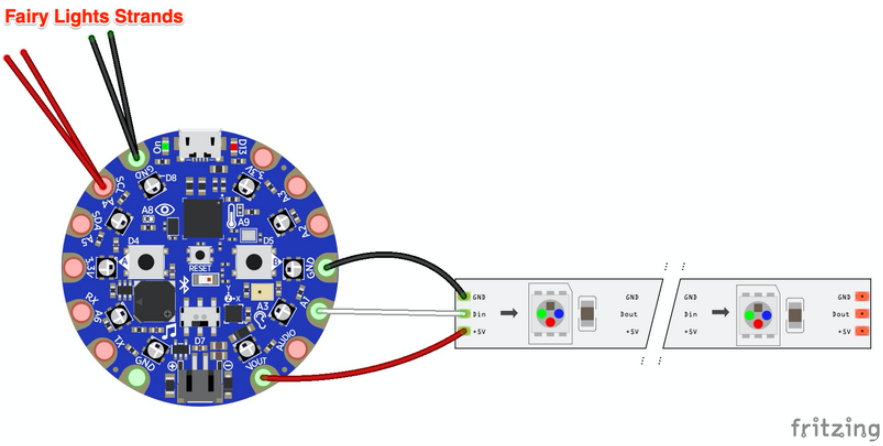
1 x Battery — https://www.adafruit.com/product/258

Lithium Ion Polymer Battery - 3.7v 1200mAh

You'll Also Need

- Yellow craft foam
- Green craft felt
- Black vinyl for the flower faces (or draw them on with a sharpie)
- Green pipe cleaners
- A baby mobile kit (https://adafru.it/RFV) or a small wooden hoop and some string
- Silk leaves or flowers for decoration

Wiring Diagram



NeoPixel strips have 3 pads for wires. Power (**+5v**) and Ground (**GND**) can be connected from either end of the strip. The middle pin (**Din** or **Dout)** must be connected from the correct end or the pixels won't light up. Always look to be sure you're connecting to **IN** for your data line.

Our NeoPixel strip will connect at its **IN** end as follows:

- **VOUT --> 5v**
- **A1 --> DIN**
- **GND --> GND**

The Fairy Lights strand wires are connected to pins **A4** and **GND** on the Circuit Playground Bluefruit.

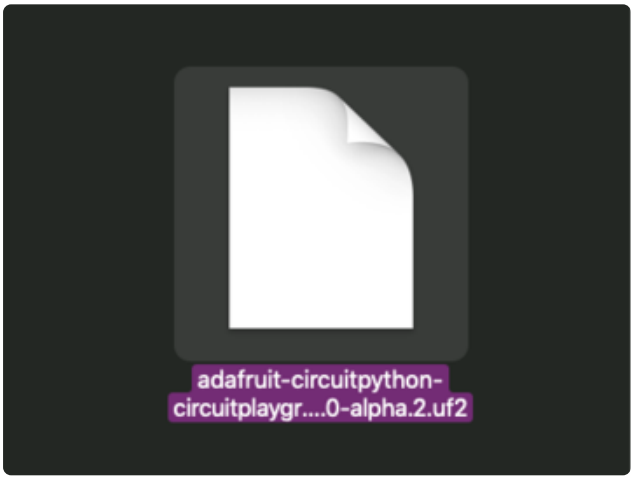
CircuitPython on Circuit Playground Bluefruit

Install or Update CircuitPython

Follow this quick step-by-step to install or update CircuitPython on your Circuit Playground Bluefruit.

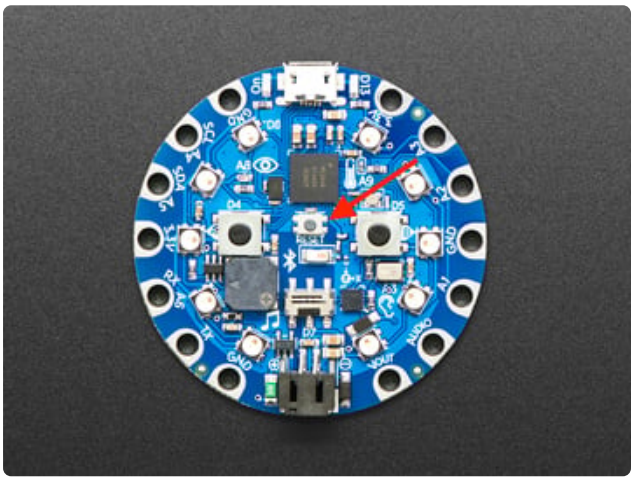
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/FNK



Click the link above and download the latest UF2 file

Download and save it to your Desktop (or wherever is handy)



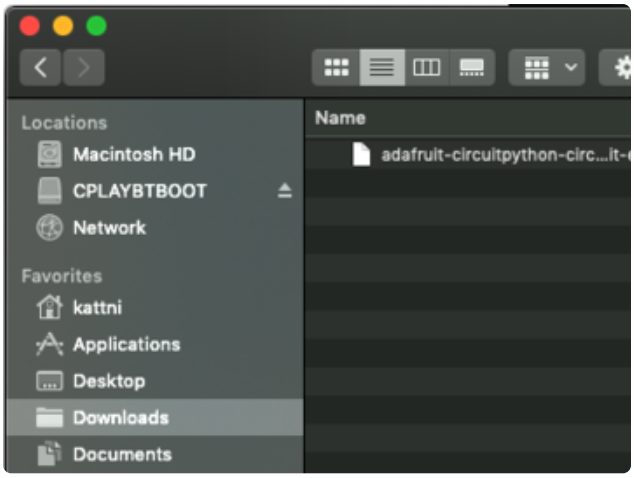
Plug your Circuit Playground Bluefruit into your computer using a known-good data-capable USB cable.

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

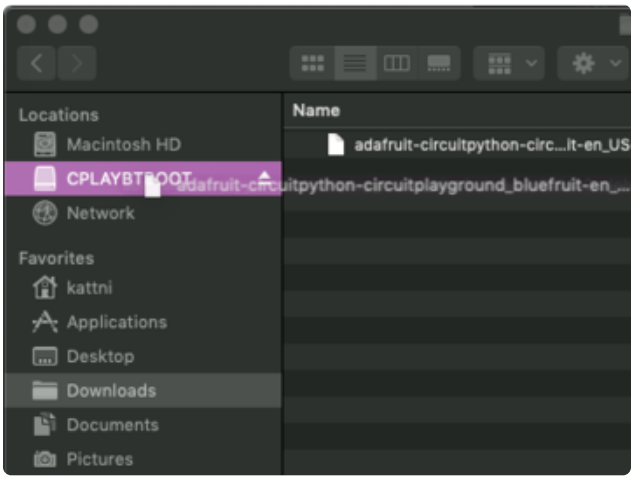
Double-click the small **Reset** button in the middle of the CPB (indicated by the red arrow in the image). The ten NeoPixel LEDs will all turn red, and then will all turn green. If they turn all red and stay red, check the USB cable, try another USB port, etc. The little red LED next to the USB connector will pulse red - this is ok!

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

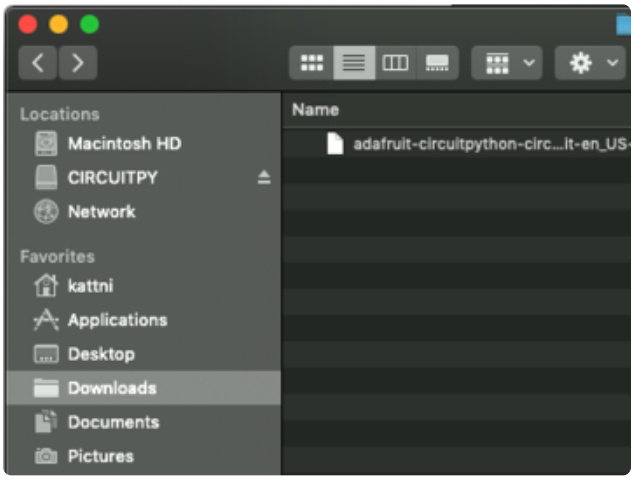
(If double-clicking doesn't do it, try a single-click!)



You will see a new disk drive appear called **CPLAYBTBOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **CPLAYBTBOOT.**



The LEDs will turn red. Then, the **CPLAYBTBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

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

Software

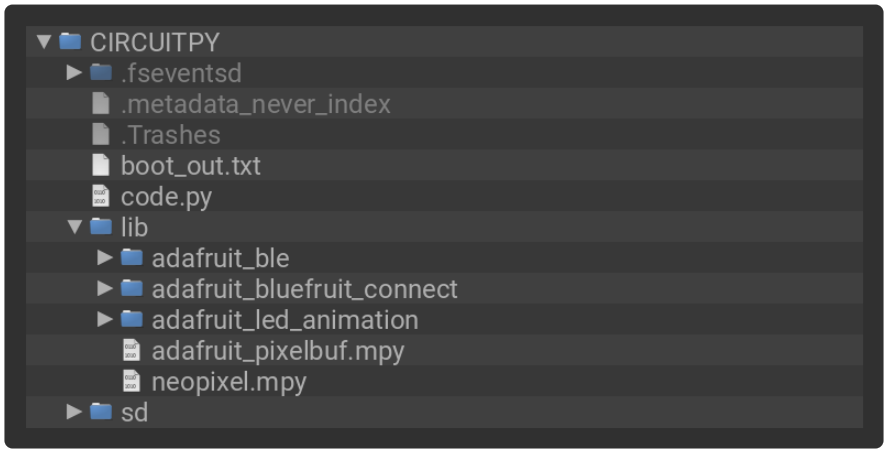
This code uses the Adafruit Bluefruit app's Control Pad and Color Picker features. The Color Picker will send a solid color to the mobile, and the Control Pad will allow you to cycle between the modes or set up quick-select buttons for your favorite modes. You can also control the brightness with the up and down arrow buttons.

Installing Project Code

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **Sunflower_Mobile/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:




```
# SPDX-FileCopyrightText: 2021 Erin St Blaine for Adafruit Industries
# SPDX-FileCopyrightText: 2021 Dan Halbert for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
LED Sunflower Mobile with Circuit Playground Bluefruit
Full tutorial:
https://learn.adafruit.com/sound-reactive-sunflower-baby-crib-mobile-with-bluetooth-
control
Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!
Written by Erin St Blaine & Dan Halbert for Adafruit Industries
Copyright (c) 2020-2021 Adafruit Industries
Licensed under the MIT license.
All text above must be included in any redistribution.

"""

import array
import math
import audiobusio
import board
import neopixel
from digitalio import DigitalInOut, Direction, Pull

from adafruit_bluefruit_connect.packet import Packet
from adafruit_bluefruit_connect.button_packet import ButtonPacket
```

```
from adafruit_bluefruit_connect.color_packet import ColorPacket
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService

from adafruit_led_animation.helper import PixelMap
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.group import AnimationGroup
from adafruit_led_animation.animation.sparkle import Sparkle
from adafruit_led_animation.animation.rainbow import Rainbow
from adafruit_led_animation.animation.rainbowchase import RainbowChase
from adafruit_led_animation.animation.rainbowcomet import RainbowComet
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.color import colorwheel
from adafruit_led_animation.color import (
    BLACK,
    RED,
    ORANGE,
    BLUE,
    PURPLE,
    WHITE,
)

YELLOW = (25, 15, 0)

# Setup BLE
ble = BLERadio()
uart = UARTService()
advertisement = ProvideServicesAdvertisement(uart)

# Color of the peak pixel.
PEAK_COLOR = (100, 0, 255)
# Number of total pixels - 10 build into Circuit Playground
NUM_PIXELS = 30


fairylights = DigitalInOut(board.A4)
fairylights.direction = Direction.OUTPUT
fairylights.value = True

# Exponential scaling factor.
# Should probably be in range -10 .. 10 to be reasonable.
CURVE = 2
SCALE_EXPONENT = math.pow(10, CURVE * -0.1)

# Number of samples to read at once.
NUM_SAMPLES = 160

brightness_increment = 0

# Restrict value to be between floor and ceiling.
def constrain(value, floor, ceiling):
    return max(floor, min(value, ceiling))


# Scale input_value between output_min and output_max, exponentially.
def log_scale(input_value, input_min, input_max, output_min, output_max):
    normalized_input_value = (input_value - input_min) / \
                             (input_max - input_min)
    return output_min + \
        math.pow(normalized_input_value, SCALE_EXPONENT) \
        * (output_max - output_min)


# Remove DC bias before computing RMS.
def normalized_rms(values):
    minbuf = int(mean(values))
```

```
    samples_sum = sum(
        float(sample - minbuf) * (sample - minbuf)
        for sample in values
    )

    return math.sqrt(samples_sum / len(values))


def mean(values):
    return sum(values) / len(values)


def volume_color(volume):
    return 200, volume * (255 // NUM_PIXELS), 0


# Main program

# Set up NeoPixels and turn them all off.
pixels = neopixel.NeoPixel(board.A1, NUM_PIXELS, brightness=0.1, auto_write=False)
pixels.fill(0)
pixels.show()

mic = audiobusio.PDMIn(board.MICROPHONE_CLOCK, board.MICROPHONE_DATA,
                       sample_rate=16000, bit_depth=16)

# Record an initial sample to calibrate. Assume it's quiet when we start.
samples = array.array('H', [0] * NUM_SAMPLES)
mic.record(samples, len(samples))
# Set lowest level to expect, plus a little.
input_floor = normalized_rms(samples) + 30
# OR: used a fixed floor
# input_floor = 50

# You might want to print the input_floor to help adjust other values.
print(input_floor)

# Corresponds to sensitivity: lower means more pixels light up with lower sound
# Adjust this as you see fit.
input_ceiling = input_floor + 100

peak = 0

# Cusomize LED Animations  -------------------------------------------------------
rainbow = Rainbow(pixels, speed=0, period=6, name="rainbow", step=2.4)
rainbow_chase = RainbowChase(pixels, speed=0.1, size=5, spacing=5, step=5)
chase = Chase(pixels, speed=0.2, color=ORANGE, size=2, spacing=6)
rainbow_comet = RainbowComet(pixels, speed=0.1, tail_length=30, bounce=True)
rainbow_comet2 = RainbowComet(
    pixels, speed=0.1, tail_length=104, colorwheel_offset=80, bounce=True
    )
rainbow_comet3 = RainbowComet(
    pixels, speed=0, tail_length=25, colorwheel_offset=80, step=4, bounce=False
    )
strum = RainbowComet(
    pixels, speed=0.1, tail_length=25, bounce=False, colorwheel_offset=50, step=4
    )
sparkle = Sparkle(pixels, speed=0.1, color=BLUE, num_sparkles=10)
sparkle2 = Sparkle(pixels, speed=0.5, color=PURPLE, num_sparkles=4)
off = Solid(pixels, color=BLACK)

# Animations Playlist - reorder as desired. AnimationGroups play at the same time
animations = AnimationSequence(

    rainbow_comet2, #
    rainbow_comet, #
    chase, #
    rainbow_chase, #
    rainbow, #
```

```
    AnimationGroup(
        sparkle,
        strum,
        ),
    AnimationGroup(
        sparkle2,
        rainbow_comet3,
        ),
    off,
    auto_clear=True,
    auto_reset=True,
)


MODE = 1
LASTMODE = 1 # start up in sound reactive mode
i = 0
# Are we already advertising?
advertising = False

while True:
    animations.animate()
    if not ble.connected and not advertising:
        ble.start_advertising(advertisement)
        advertising = True

    # Are we connected via Bluetooth now?
    if ble.connected:
        # Once we're connected, we're not advertising any more.
        advertising = False
        # Have we started to receive a packet?
        if uart.in_waiting:
            packet = Packet.from_stream(uart)
            if isinstance(packet, ColorPacket):
                # Set all the pixels to one color and stay there.
                pixels.fill(packet.color)
                pixels.show()
                MODE = 2
            elif isinstance(packet, ButtonPacket):
                if packet.pressed:
                    if packet.button == ButtonPacket.BUTTON_1:
                        animations.activate(1)
                    elif packet.button == ButtonPacket.BUTTON_2:
                        MODE = 1
                        animations.activate(2)
                    elif packet.button == ButtonPacket.BUTTON_3:
                        MODE = 1
                        animations.activate(3)
                    elif packet.button == ButtonPacket.BUTTON_4:
                        MODE = 4

                    elif packet.button == ButtonPacket.UP:
                        pixels.brightness = pixels.brightness + 0.1
                        pixels.show()
                        if pixels.brightness > 1:
                            pixels.brightness = 1
                    elif packet.button == ButtonPacket.DOWN:
                        pixels.brightness = pixels.brightness - 0.1
                        pixels.show()
                        if pixels.brightness < 0.1:
                            pixels.brightness = 0.1
                    elif packet.button == ButtonPacket.RIGHT:
                        MODE = 1
                        animations.next()
                    elif packet.button == ButtonPacket.LEFT:
                        animations.activate(7)
            animations.animate()
```

```
        if MODE == 2:
            animations.freeze()
        if MODE == 4:
            animations.freeze()
            pixels.fill(YELLOW)
            mic.record(samples, len(samples))
            magnitude = normalized_rms(samples)
            # You might want to print this to see the values.
            #print(magnitude)

            # Compute scaled logarithmic reading in the range 0 to NUM_PIXELS
            c = log_scale(constrain(magnitude, input_floor, input_ceiling),
                          input_floor, input_ceiling, 0, NUM_PIXELS)

            # Light up pixels that are below the scaled and interpolated magnitude.
            #pixels.fill(0)
            for i in range(NUM_PIXELS):
                if i < c:
                    pixels[i] = volume_color(i)
                # Light up the peak pixel and animate it slowly dropping.
                if c >= peak:
                    peak = min(c, NUM_PIXELS - 1)
                elif peak > 0:
                    peak = peak - 0.01
                if peak > 0:
                    pixels[int(peak)] = PEAK_COLOR
            pixels.show()
```

Code Walkthrough

Code Walkthrough

First we import all the library elements needed.

```
import array
import math
import audiobusio
import board
import neopixel
from digitalio import DigitalInOut, Direction, Pull

from adafruit_bluefruit_connect.packet import Packet
from adafruit_bluefruit_connect.button_packet import ButtonPacket
from adafruit_bluefruit_connect.color_packet import ColorPacket
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService

from adafruit_led_animation.helper import PixelMap
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.group import AnimationGroup
from adafruit_led_animation.animation.sparkle import Sparkle
from adafruit_led_animation.animation.rainbow import Rainbow
from adafruit_led_animation.animation.rainbowchase import RainbowChase
from adafruit_led_animation.animation.rainbowcomet import RainbowComet
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.color import colorwheel
from adafruit_led_animation.color import (
    BLACK,
    RED,
```

```
    ORANGE,
    BLUE,
    PURPLE,
    WHITE,
)
```

The CircuitPython LED Animations Library will do most of the heavy lifting for the actual pixel animations. I've imported most of the basic colors I want to use. It's also easy to define or redefine your own colors. I wanted a dim yellow to use as the background in the sound reactive mode, so defined YELLOW with an RGB tuple:

```
YELLOW = (25, 15, 0)
```

Next we'll set up the bluetooth service and define some variables. It's hard to tell exactly how many pixels we have in this uber-high-density strip, but there appear to be around 30.

We'll also define the fairy light strands on pin A4.

```
# Setup BLE
ble = BLERadio()
uart = UARTService()
advertisement = ProvideServicesAdvertisement(uart)

# Color of the peak pixel.
PEAK_COLOR = (100, 0, 255)
# Number of total pixels - 10 build into Circuit Playground
NUM_PIXELS = 30


fairylights = DigitalInOut(board.A4)
fairylights.direction = Direction.OUTPUT
fairylights.value = True
```

Next we'll set up the sound reactive code. This function is based on the Playground Sound Meter guide. (https://adafru.it/BXq)

```
# Exponential scaling factor.
# Should probably be in range -10 .. 10 to be reasonable.
CURVE = 2
SCALE_EXPONENT = math.pow(10, CURVE * -0.1)

# Number of samples to read at once.
NUM_SAMPLES = 160

brightness_increment = 0

# Restrict value to be between floor and ceiling.
def constrain(value, floor, ceiling):
    return max(floor, min(value, ceiling))


# Scale input_value between output_min and output_max, exponentially.
def log_scale(input_value, input_min, input_max, output_min, output_max):
    normalized_input_value = (input_value - input_min) / \
                             (input_max - input_min)
    return output_min + \
```

```
        math.pow(normalized_input_value, SCALE_EXPONENT) \
        * (output_max - output_min)


# Remove DC bias before computing RMS.
def normalized_rms(values):
    minbuf = int(mean(values))
    samples_sum = sum(
        float(sample - minbuf) * (sample - minbuf)
        for sample in values
    )

    return math.sqrt(samples_sum / len(values))


def mean(values):
    return sum(values) / len(values)


def volume_color(volume):
    return 200, volume * (255 // NUM_PIXELS), 0
```

Next, we set up our NeoPixel object on pin A1 and set them to "off" initially.

```
pixels = neopixel.NeoPixel(board.A1, NUM_PIXELS, brightness=0.1, auto_write=False)
pixels.fill(0)
pixels.show()
```

Then, set up the microphone input for our sound reaction.

```
mic = audiobusio.PDMIn(board.MICROPHONE_CLOCK, board.MICROPHONE_DATA,
                       sample_rate=16000, bit_depth=16)

# Record an initial sample to calibrate. Assume it's quiet when we start.
samples = array.array('H', [0] * NUM_SAMPLES)
mic.record(samples, len(samples))
# Set lowest level to expect, plus a little.
input_floor = normalized_rms(samples) + 30
# OR: used a fixed floor
# input_floor = 50

# You might want to print the input_floor to help adjust other values.
print(input_floor)

# Corresponds to sensitivity: lower means more pixels light up with lower sound
# Adjust this as you see fit.
input_ceiling = input_floor + 100

peak = 0
```

Next we define our LED animations. You can learn more about how these animations work and which type of animations are available in the CircuitPython LED Animations guide (https://adafru.it/LZF).

We've set up a handful of animations and then put them into an animation playlist. Later on in the code we can refer to these animations by number and assign them to different buttons in the Bluetooth control app.

Animations can also be layered using AnimationGroup. The LED Animations library is really powerful and easy to use, so dig in and customize here to your heart's content.

```
# Cusomize LED Animations  -------------------------------------------------------
rainbow = Rainbow(pixels, speed=0, period=6, name="rainbow", step=2.4)
rainbow_chase = RainbowChase(pixels, speed=0.1, size=5, spacing=5, step=5)
chase = Chase(pixels, speed=0.2, color=ORANGE, size=2, spacing=6)
rainbow_comet = RainbowComet(pixels, speed=0.1, tail_length=30, bounce=True)
rainbow_comet2 = RainbowComet(
    pixels, speed=0.1, tail_length=104, colorwheel_offset=80, bounce=True
    )
rainbow_comet3 = RainbowComet(
    pixels, speed=0, tail_length=25, colorwheel_offset=80, step=4, bounce=False
    )
strum = RainbowComet(
    pixels, speed=0.1, tail_length=25, bounce=False, colorwheel_offset=50, step=4
    )
sparkle = Sparkle(pixels, speed=0.1, color=BLUE, num_sparkles=10)
sparkle2 = Sparkle(pixels, speed=0.5, color=PURPLE, num_sparkles=4)
off = Solid(pixels, color=BLACK)

# Animations Playlist - reorder as desired. AnimationGroups play at the same time
animations = AnimationSequence(

    rainbow_comet2, #
    rainbow_comet, #
    chase, #
    rainbow_chase, #
    rainbow, #

    AnimationGroup(
        sparkle,
        strum,
        ),
    AnimationGroup(
        sparkle2,
        rainbow_comet3,
        ),
    off,
    auto_clear=True,
    auto_reset=True,
)


MODE = 1
LASTMODE = 1 # start up in sound reactive mode
i = 0
# Are we already advertising?
advertising = False
```

Finally, we get to the main program loop. The pixels will turn on and play the first animation in the Animation Sequence when they're powered up. The bluetooth will begin advertising.

This is the area where you can define which modes play when you press each of the buttons in the Bluetooth Control App. Just change the number for each mode button called by `animations.animate()`.

Mode 4 turns on Sound Reactive mode, and I've assigned button 4 to activate it.

```
while True:
    animations.animate()
    if not ble.connected and not advertising:
        ble.start_advertising(advertisement)
        advertising = True

    # Are we connected via Bluetooth now?
    if ble.connected:
        # Once we're connected, we're not advertising any more.
        advertising = False
        # Have we started to receive a packet?
        if uart.in_waiting:
            packet = Packet.from_stream(uart)
            if isinstance(packet, ColorPacket):
                # Set all the pixels to one color and stay there.
                pixels.fill(packet.color)
                pixels.show()
                MODE = 2
            elif isinstance(packet, ButtonPacket):
                if packet.pressed:
                    if packet.button == ButtonPacket.BUTTON_1:
                        animations.activate(1)
                    elif packet.button == ButtonPacket.BUTTON_2:
                        MODE = 1
                        animations.activate(2)
                    elif packet.button == ButtonPacket.BUTTON_3:
                        MODE = 1
                        animations.activate(3)
                    elif packet.button == ButtonPacket.BUTTON_4:
                        MODE = 4

                    elif packet.button == ButtonPacket.UP:
                        pixels.brightness = pixels.brightness + 0.1
                        pixels.show()
                        if pixels.brightness &gt; 1:
                            pixels.brightness = 1
                    elif packet.button == ButtonPacket.DOWN:
                        pixels.brightness = pixels.brightness - 0.1
                        pixels.show()
                        if pixels.brightness &lt; 0.1:
                            pixels.brightness = 0.1
                    elif packet.button == ButtonPacket.RIGHT:
                        MODE = 1
                        animations.next()
                    elif packet.button == ButtonPacket.LEFT:
                        animations.activate(7)
            animations.animate()

        if MODE == 2:
            animations.freeze()
        if MODE == 4:
            animations.freeze()
            pixels.fill(YELLOW)
            mic.record(samples, len(samples))
            magnitude = normalized_rms(samples)
            # You might want to print this to see the values.
            #print(magnitude)

            # Compute scaled logarithmic reading in the range 0 to NUM_PIXELS
            c = log_scale(constrain(magnitude, input_floor, input_ceiling),
                          input_floor, input_ceiling, 0, NUM_PIXELS)

            # Light up pixels that are below the scaled and interpolated magnitude.
            #pixels.fill(0)
            for i in range(NUM_PIXELS):
                if i &lt; c:
                    pixels[i] = volume_color(i)
                # Light up the peak pixel and animate it slowly dropping.
```

```
            if c &gt;= peak:
                peak = min(c, NUM_PIXELS - 1)
            elif peak &gt; 0:
                peak = peak - 0.01
            if peak &gt; 0:
                pixels[int(peak)] = PEAK_COLOR
        pixels.show()
```

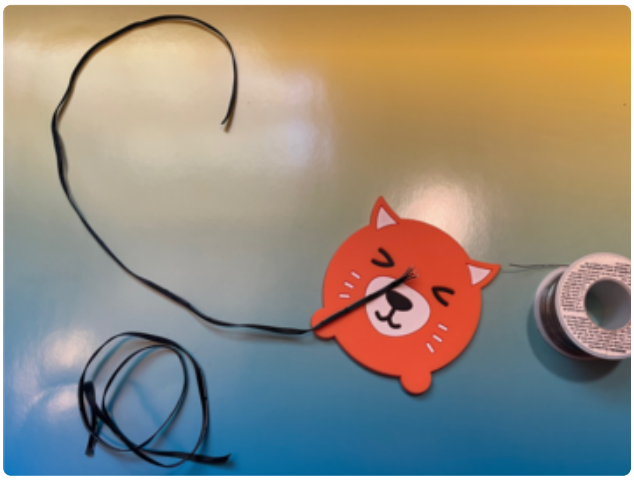
Troubleshooting

If you're having trouble getting the code to load, head over to the Circuit Playground Bluefruit (https://adafru.it/GYc) guide for more detailed instructions and things to try.

The Mu Editor has a very handy feature called the REPL, which will help you with debugging and give you feedback about what may be going wrong. Click the Serial button in the toolbar to access the REPL. Here's a lot more info about this process (https://adafru.it/RPC) -- it's invaluable in debugging your code.

Electronics Assembly

Trim both ends off your LED strip and pull it out of the silicone casing. Flip the strip over so you're looking at the back. Count out 6 copper pads from one end, and carefully cut through the middle of the pads.



Prepare your ribbon cable. For my project I estimated needing about 1/2m of 3-wire ribbon for each sunflower. Our 30awg wire ribbon is 1 m and has 4 wires, so, I cut the ribbon cable in half and stripped off one wire (do the one on the edge that is NOT striped).

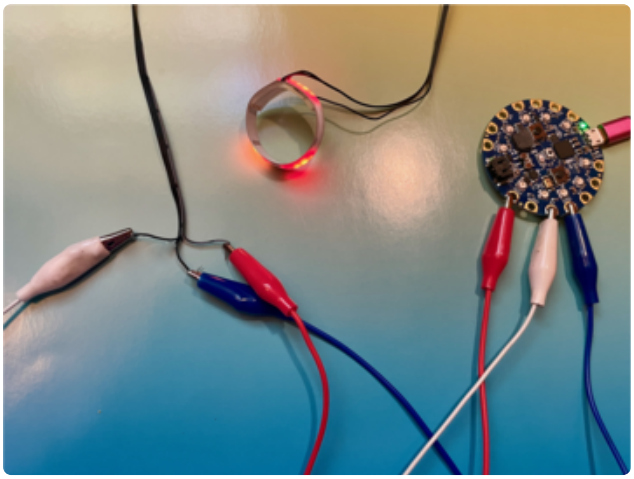
Strip about 1/8" of shielding from each of the 3 wires and tin the ends of the wire.

Find the "in" end of the strip. For me, it's the end on the left if the writing on the front of the strip is right-side up.

Tin the pads on this end of the strip. Tinning both the wires and pads ahead of time will make it much easier to solder to the tiny pads -- all you need to do is re-melt the solder with the tip of the soldering iron to get a good connection.

Solder the striped wire to the pad marked **5V**, the middle wire to the middle (**Din**) pad, and the remaining wire to the **G** pin.

Bend your strip around in a ring with the LED pixels facing outward. Use a piece of strong tape on the inside of the ring to secure the ends into a circle. Make sure the copper pads aren't touching or overlapping on the two ends of the strip.



Connect your alligator clips to the other end of the wire ribbon: red to striped, white to middle, and black to the remaining wire. Hook the alligator clips up to your Circuit Playground Bluefruit (once you've loaded the software) and test to be sure your ring lights up.

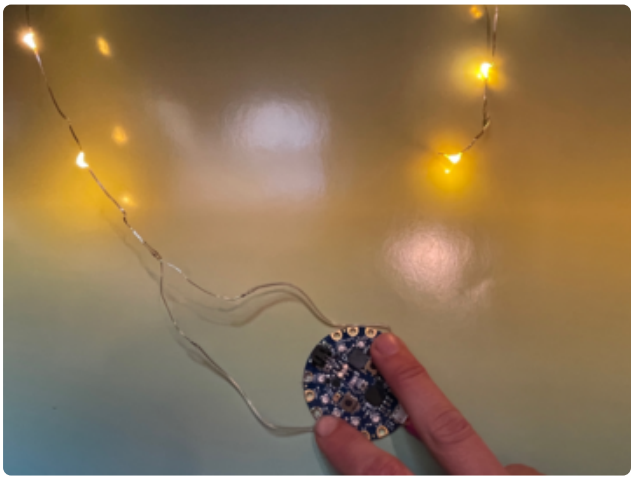
Make a ring for each of your sunflowers. My mobile has 5 sunflowers, which just about uses up my 1/2m strip of LEDs with a few extras left over in case I'm sloppy with my cuts.

Set the wired rings aside for now. We'll make the all flowers before soldering the other ends of the wire permanently to the Circuit Playground Bluefruit.

Fairy Lights

I used two strands of warm white fairy lights to add more interest and dimension to my mobile.

Cut the battery pack apart from the wire strand. If you cut through the lead wire you'll have a much easier time than if you cut through the copper wire closer to the lights.



Strip off about 1/4" of shielding from both wires. One of these wires will go to a G pin, and the other will go to pin A4. Using a digital in/out pin as our power connection will allow us to turn the fairy lights on and off in the software. Cool!

For testing purposes, gently press one of the bare wires to the 3v pad and the other to G. If the strand lights up, you've got them oriented right. If it doesn't, switch the wires around until you know which wire is which.

Repeat with any additional fairy lights strands -- my mobile uses 2 strands. Solder the power wire(s) from your strands to pin **A4**, and your ground wire(s) to **G** on the Circuit Playground.

Make the Sunflowers

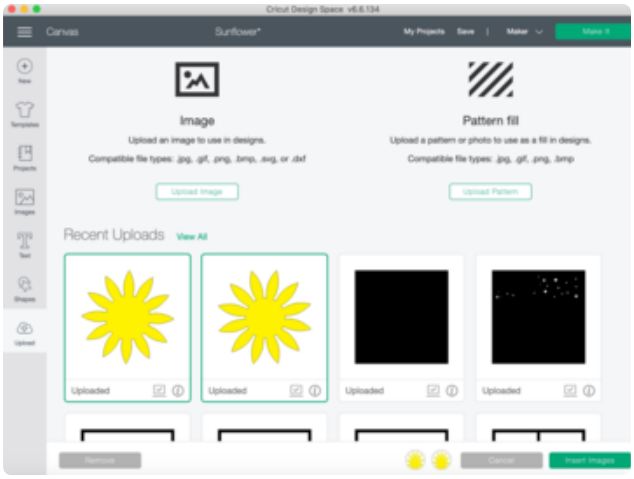
These sunflower shapes can be cut by hand with scissors or a utility knife, or can be cut on a vinyl cutting machine. I'm using a Cricut Maker machine with a strong grip mat and a variety of cutting blades including the knife blade and the rotary cutting blade.

The flower shapes are made from EVA craft foam sheets in various shades of yellow, and the leaves are made from acrylic felt in green. This stuff is really inexpensive and easy to find at any craft store.

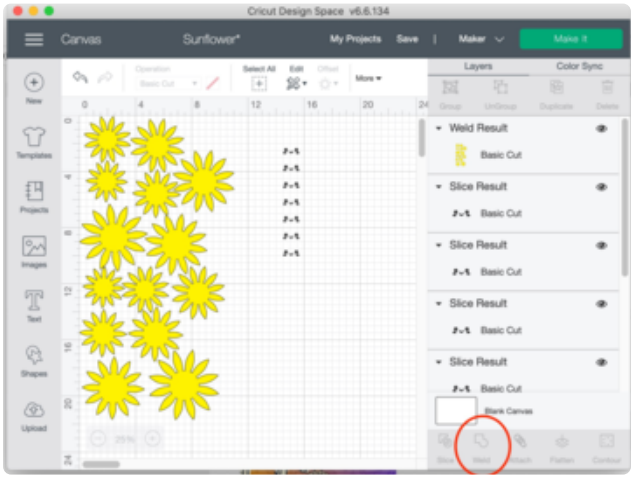
Download the files below. The .png files are best for printing and using as a hand-cut template, and the .svg files are best for uploading to your vinyl cutter's Design software.

sunflower_files.zip

https://adafru.it/RFU



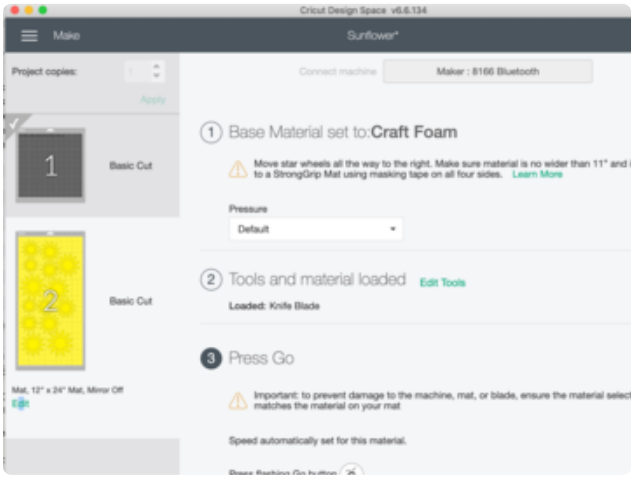
You'll find two slightly different sunflower files and a file for the face graphic. Upload all three .svg files to the design software for your vinyl cutter and save them as "cut" images.



I wanted a variety of sizes for my flowers. After inserting the images, duplicate them as many times as you'd like and resize them so they're the size you want. My smallest flowers are around 2" across and my largest are around 4".

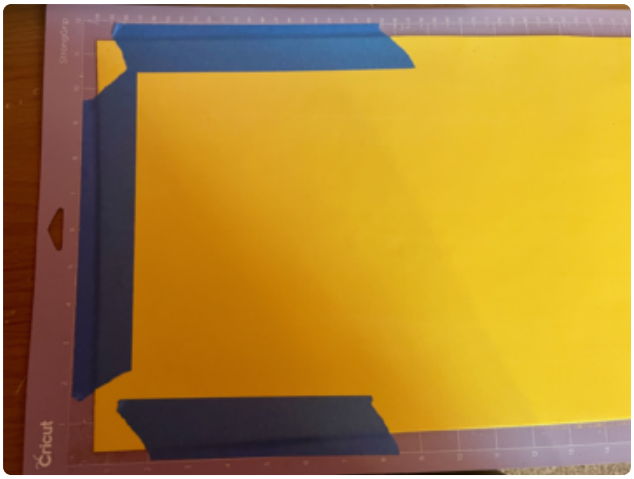
Remember you'll be hiding the NeoPixel ring you made in the center, so don't make them so small that it will stick out.

Duplicate the face image as well so there are enough for all the flowers you're making.

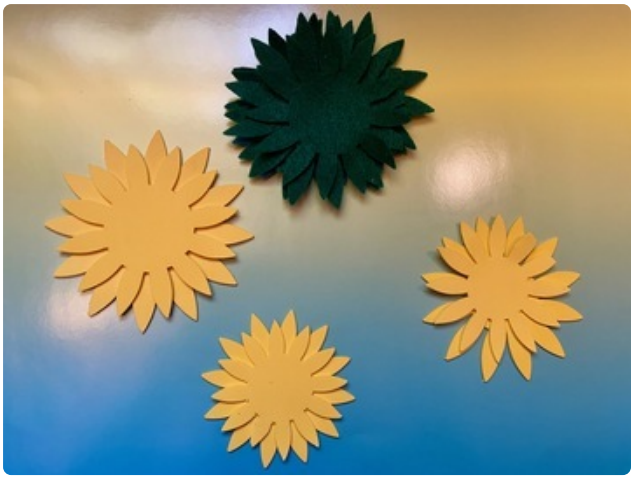


Once you're happy, click "Make It". Choose the correct material and cutting blade. I had success by choosing "craft foam" along with the knife blade, and "acrylic felt" along with the rotary cutting blade on my Cricut Maker.

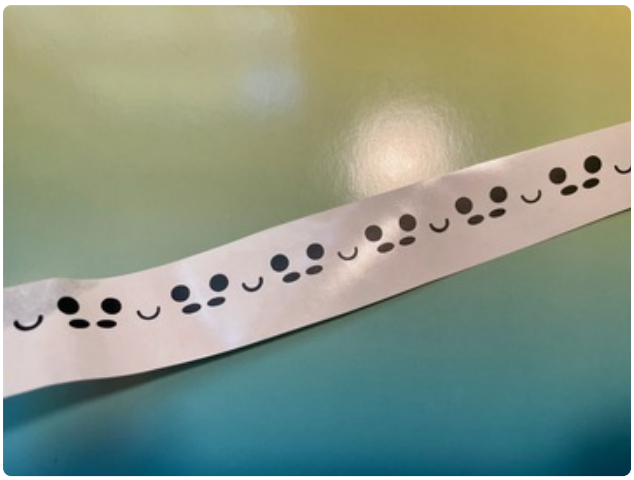
This screen is also where you can arrange your flowers so they will fit on your material. Nudge them in from the edges a bit since our material won't go all the way to the edge of the mat.



Trim 1" off your craft foam so it's 11 inches wide. Secure it to your strong grip mat with tape on all 4 sides to keep it from moving around while you're cutting.



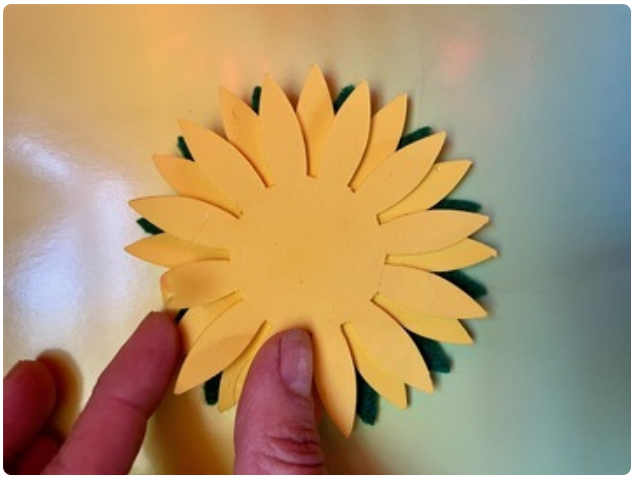
Cut your flowers. Each flower will have three layers: sunflower1, sunflower2, and another layer (either sunflower1 or 2) cut from green felt for the leaves.



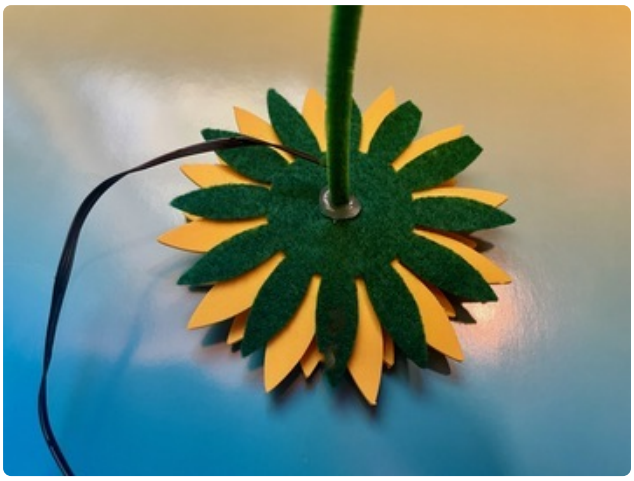
Cut the face stickers from black matte vinyl. It may also be fun to personalize your flowers with hand-drawn faces.

Use hot glue to stick your LED ring right in the center of the smaller (front) sunflower piece.

Poke a hole in the center of your back sunflower piece and your green leaf piece. Thread the wires from the ring through this hole. Glue these two pieces to the back of the NeoPixel ring once the wires are threaded through. Be sure to offset the petals so they peek between the petals of the front flower.



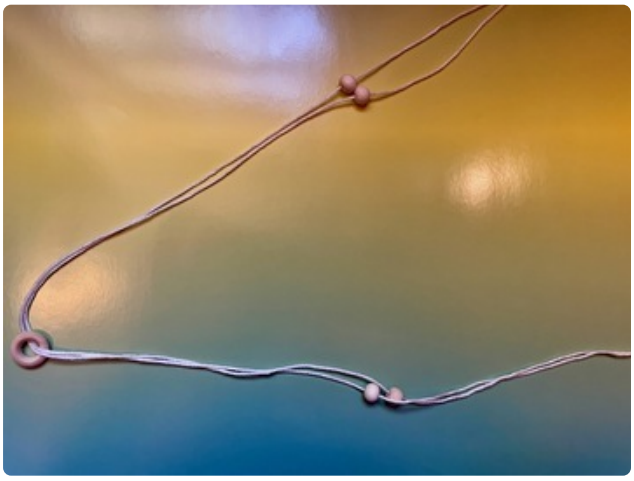
Add some character to your flowers by curling up the leaves a bit and decorate each one with a smiling face.



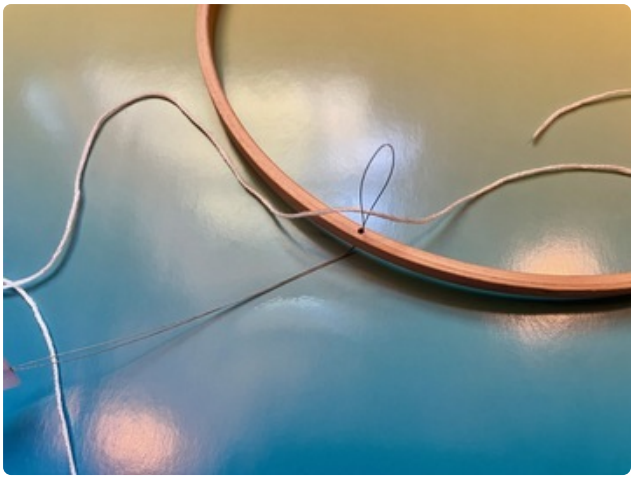
Glue a green pipe cleaner to the back of each flower to serve as a stem. Discreetly wind the wires around the stem to make them begin to fade into invisibility.

Build the Mobile

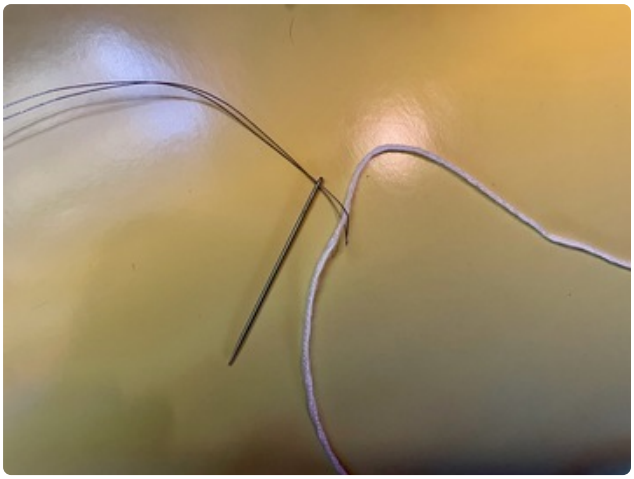
I ordered a baby mobile kit from Amazon (https://adafru.it/RFV) which worked great. It's nothing much more than a 9" wooden hoop with holes pre-drilled for the included string. This one also included some cute wooden beads, and a needle and wire needle-threader to help with assembly.



Cut two lengths of string, at least 3-4 feet long. Thread them through a ring for hanging. If you want to add beads or crystals, thread them on now.



Thread the string through the holes on the mobile. My mobile kit included a handy piece of bent wire to make threading the holes easy. Push the wire through the hole, thread the string through the wire, then pull the wire back out again, drawing the string down neatly through the hole.



To add more beads or crystals on the strings, you can thread a needle with a large eye using the same trick to make the beading easier.

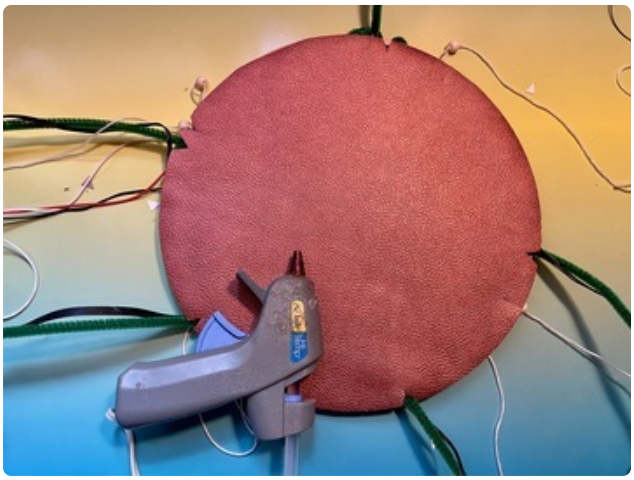
It's easiest to assemble the mobile while it's hanging. Hang it up and make sure the weight is balanced between the four strings before tying them off securely.

Wrap the ends of your flower stems around the hoop. I placed mine at varying heights, for interest and variety.

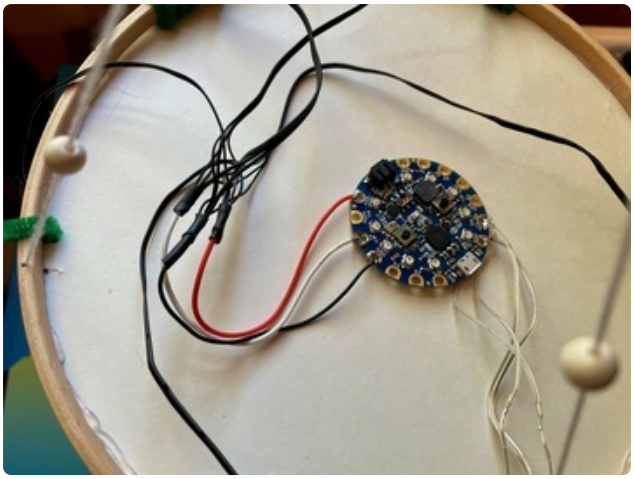
This is a great time to test again to make sure all your flowers are still working. It's much easier to fix any glitches now.

Grab your trusty alligator clips again. Strip about 1/4" from each striped wire and twist all 5 striped wires together. Attach the red alligator clip to this bundle, and clip the other end to **VOUT** on your Circuit Playground. Repeat with the middle wires and connect to **A1**, and then connect the ground wires to **G**. Make sure everything is lighting up with no flickers.

To hide the wires, cut a circle from stiff poster board or cardboard that's the same size as your hoop. I cut a few notches where the sunflowers will drape down so everything lays out evenly.



Cut a second circle from pretty vinyl or another material to hide the poster board. Glue the circles to the underside of the wooden hoop.



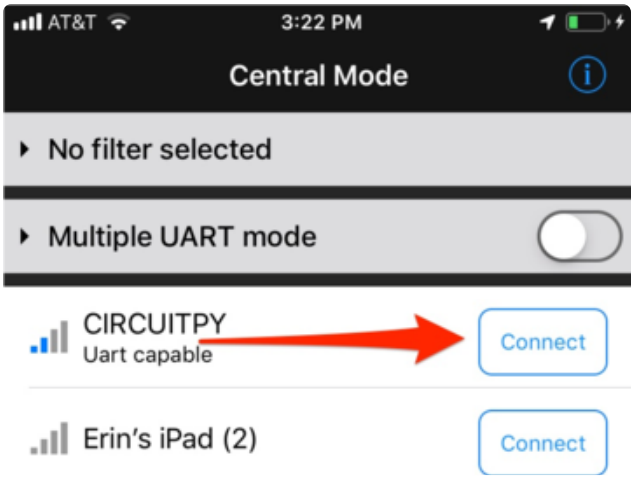
Finally, it's time to connect our sunflower wires to the Circuit Playground Bluefruit. I found it easiest to splice all the striped wire to one red wire, then connect that to **VOUT**. Splice all the data wires to a white wire and connect to **A1**, then repeat with the ground wires to **G**.

Decorate your mobile with more sunflowers, silk leaves, bees, butterflies, or any other elements that make your baby smile.

Use It

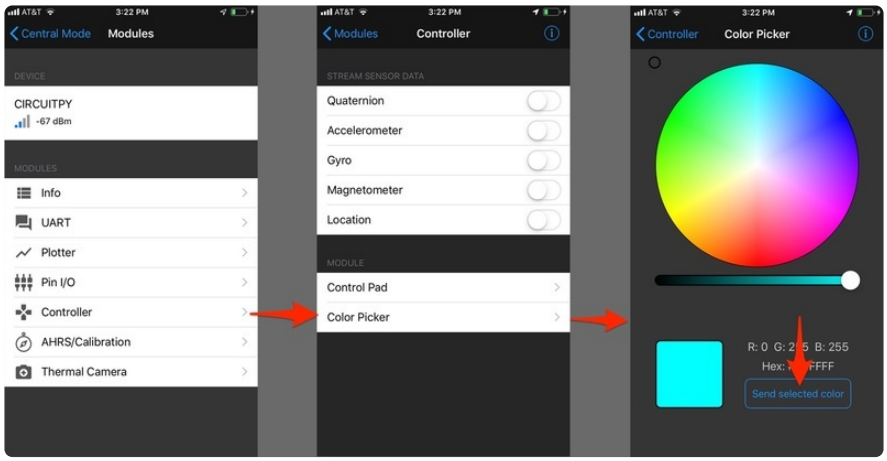
Adafruit Bluefruit App

Go to the app store (iOS or Android) and find and install the Adafruit Bluefruit app on your smartphone.

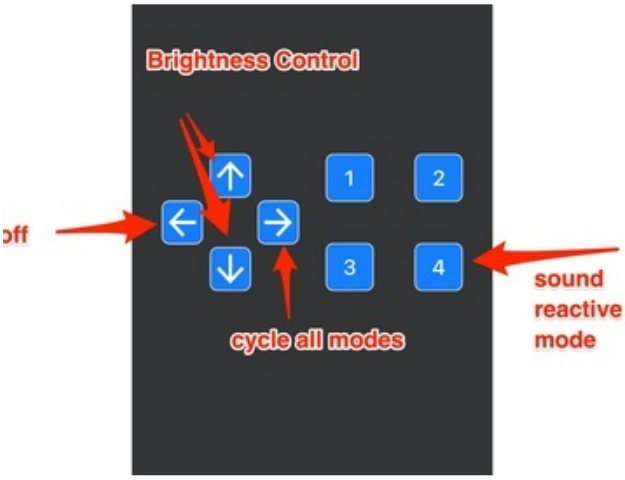


Launch the app. Make sure you're in "Central Mode" and that your Circuit Playground Bluefruit is powered on. You should see the **CIRCUITPY** drive appear on the main screen. If it's not there, try pulling down to refresh or restarting the app.

If you still don't see it, try re-uploading your code, making sure you have all the libraries installed. It won't work without all the libraries! CircuitPython still lets you save the file without throwing an error, so double check you've got everything you need.



Click Connect, then Controller > Color Picker. Choose a color and press the "Send selected color" button. Your luminaries will update with the new color. Like magic!



Now go back to the Control Pad. You'll see four buttons and four arrows. The number buttons will select your preset modes, and the up and down arrows will increase or decrease the brightness. The left arrow turns the NeoPixels off and the right arrow cycles through all your modes.

It probably goes without saying, but for posterity: the mobile should be solidly mounted, and any wiring or small, choke-able parts kept well out of reach of small children.