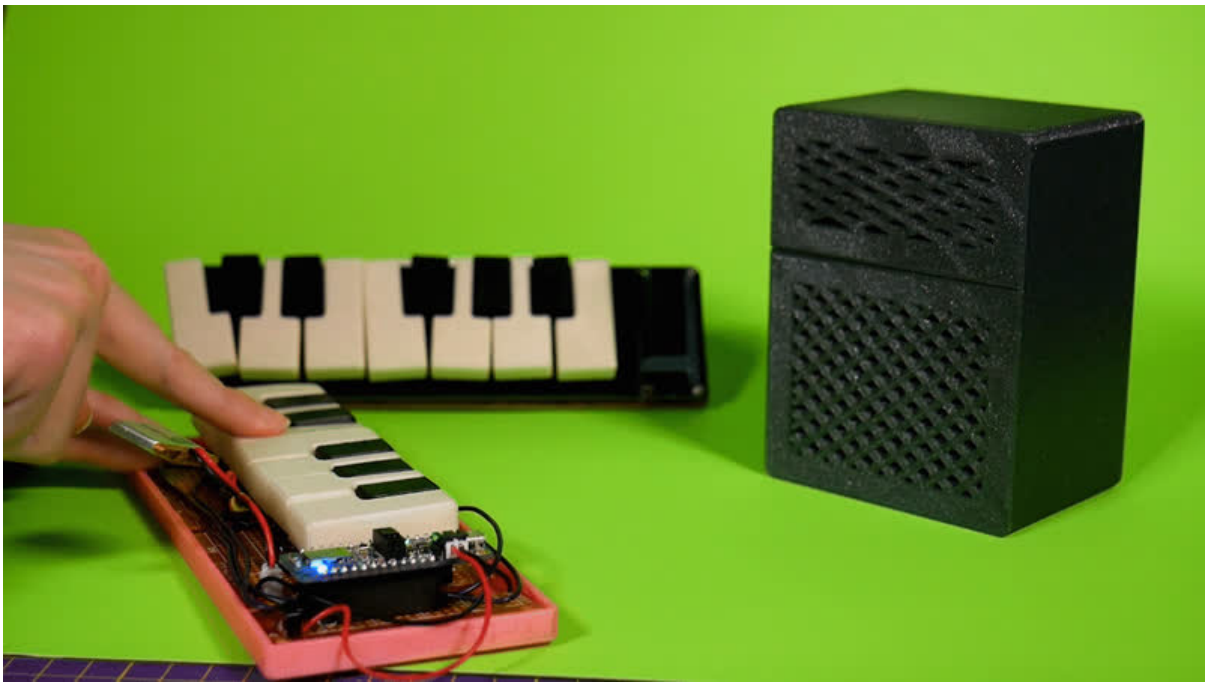
BLE Synth with the Feather nRF52840 and Circuit Playground Bluefruit

Created by Liz Clark

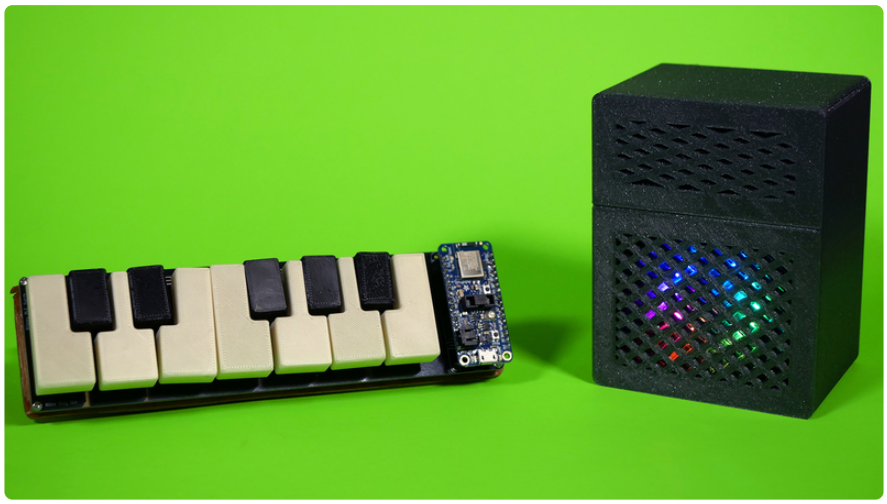


https://learn.adafruit.com/ble-synth-with-the-feather-nrf52840-and-circuit-playground-bluefruit

Last updated on 2024-06-03 03:02:32 PM EDT

Table of Contents

Overview

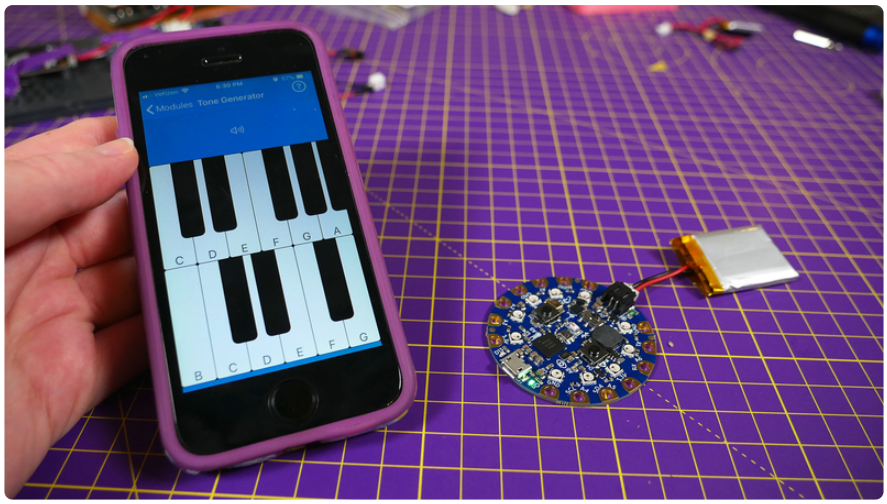


In this project, Bluetooth Low Energy (BLE) is used to have a **Feather nRF52840** and **Circuit Playground Bluefruit** communicate with each other to create a wireless 8-bit music synthesizer. The Feather has 12 buttons attached to its I/O pins. Each time a button is pressed, it sends a BLE packet to the Circuit Playground Bluefruit. When the Circuit Playground Bluefruit receives those packets, it swirls its on-board NeoPixels in a specific color and plays a tone.

The Bluefruit Playground App

This project was inspired by the tone example in the **Bluefruit Playground App**. With the app, you may connect your Circuit Playground Bluefruit board to your iOS device and then play with a series of examples, each with a user interface to allowing control of the Circuit Playground Bluefruit sensors.

One of those examples has a piano keyboard in the app and when the keys are pressed, they cause a tone to play from the Circuit Playground Bluefruit. I thought it would be cool to recreate this concept with hardware.

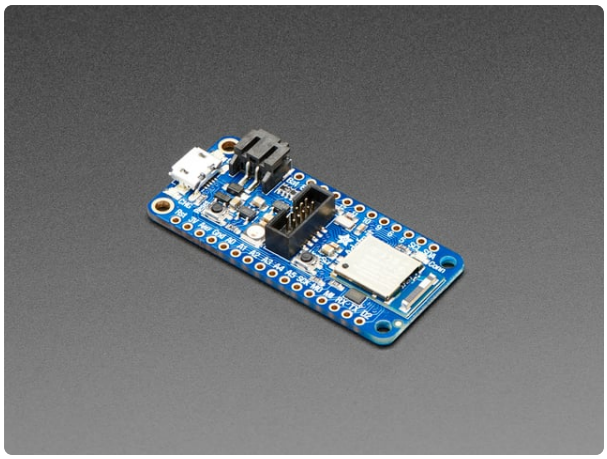


Check out this Learn Guide on the Bluefruit Playground App

https://adafru.it/HCE

Project Video

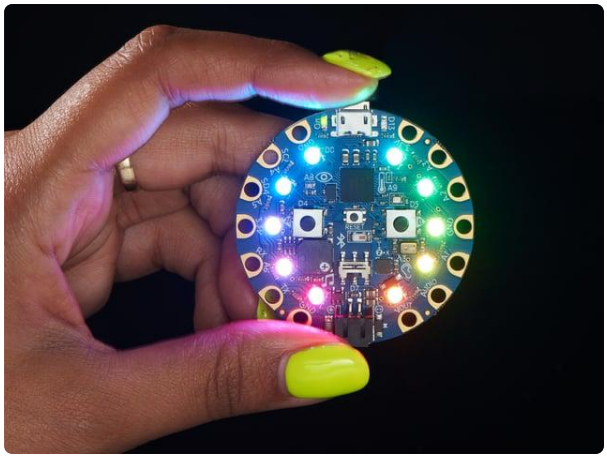
Parts



Adafruit Feather nRF52840 Express

The Adafruit Feather nRF52840 Express is the new Feather family member with Bluetooth Low Energy and native USB support featuring the nRF52840! It's...

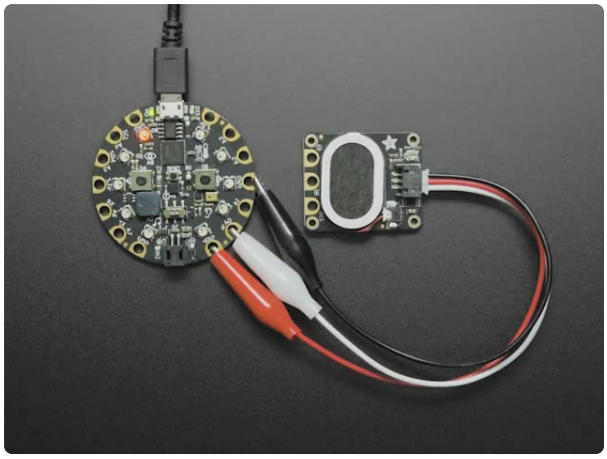
https://www.adafruit.com/product/4062



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 STEMMA Speaker - Plug and Play Audio Amplifier

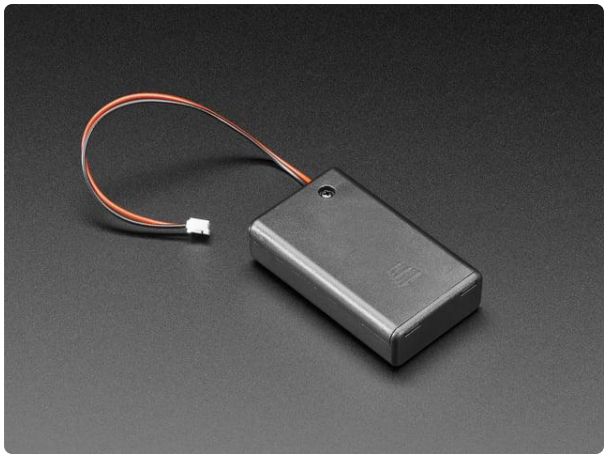
Hey, have you heard the good news? With Adafruit STEMMA boards you can easily and safely plug sensors and devices together, like this Adafruit STEMMA Speaker - Plug and Play...

https://www.adafruit.com/product/3885

Colorful Round Tactile Button Switch Assortment - 15 pack

Little clicky switches are standard input "buttons" on electronic projects. These work best in a PCB but can be...

https://www.adafruit.com/product/1009



3 x AAA Battery Holder with On/Off Switch and 2-Pin JST

This battery holder connects 3 AAA batteries together in series for powering all kinds of projects. We spec'd these out because the box is slim, and 3 AAA's add up to about...

https://www.adafruit.com/product/727

1 x Perfboard Plates — https://www.adafruit.com/product/2670

Bakelite, universal, pack of 10

---

1 x Slide Switch — https://www.adafruit.com/product/805

SPDT - breadboard friendly

---

1 x Short Female Headers for Feather — https://www.adafruit.com/product/2940

One 12-pin and one 16-pin

Helpful Tools

Super Scissors
Save your scissors! Instead of using your nice shears (and dulling them) or wire cutters (not right for the job) - use these super scissors. They're meant for engineering/maker...
https://www.adafruit.com/product/1599

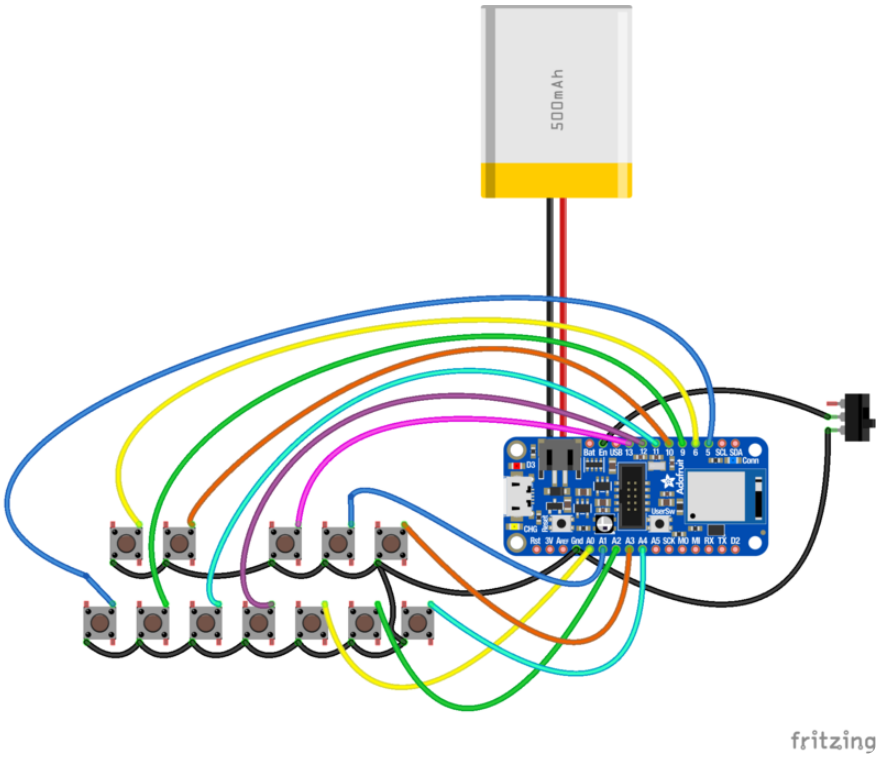
Electronics

Overall, the electronics for this project are fairly straightforward. Most of the hard work will be happening in the code.

Feather nRF52840 Keyboard

The **Feather** is acting as the **keyboard**, it has 12 buttons hooked up to its I/O pins. The Feather has just enough pins, with a few extra to spare. The buttons will represent the notes on a keyboard chromatically: C, C#, D, D#, E, F, F#, G, G#, A, A# and B.

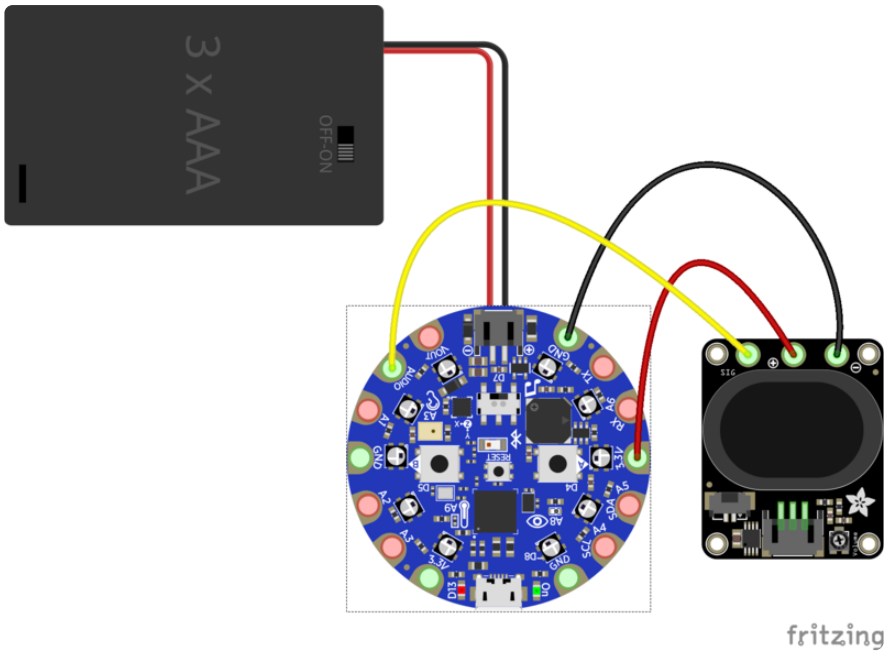
We'll also add an on/off switch by having a switch between the Feather **GND** and **EN** pins.



Circuit Playground Bluefruit Amp

The **Circuit Playground Bluefruit** is our speaker, or **amplifier**, for our 8-bit synth. It does have an onboard speaker that we can use, but for maximum loudness we're also going to add a **STEMMA Speaker**.

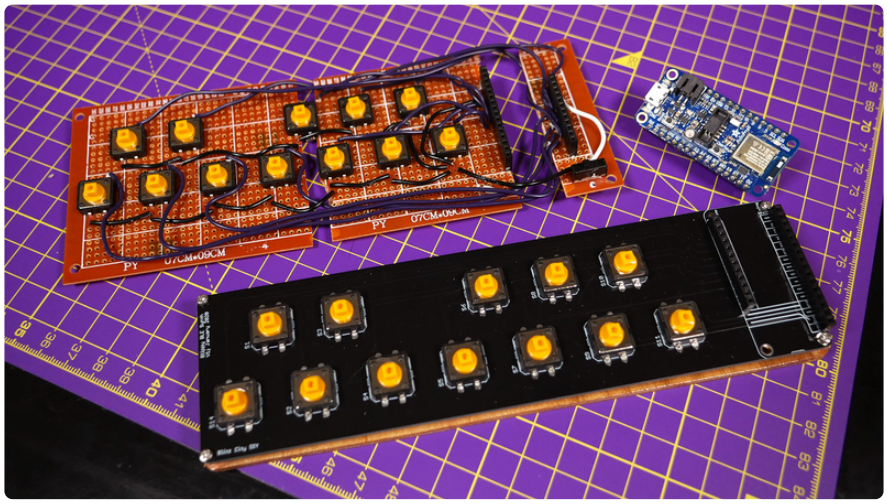
The **STEMMA Speaker** has three connections that need to be made: **POWER**, **GND** and **SIGNAL**. We'll add an on/off switch by using a **3x AAA battery pack** that plugs into the **JST battery connector** with a built-in on/off switch. This will be able to fit inside our 3D printed case.



Soldering - Feather Keyboard

PCB vs. Perfboard

A PCB was created for this project and the Eagle files are available below. If PCBs aren't your thing though, don't worry- there are instructions below on how to solder up a version using perfboard. Since the circuit is straightforward, either option is great.



PCB

For the PCB assembly, place the buttons into their spots on the board and solder each of their four points on the back.

Then, solder some **Feather headers** into the header holes. And with that you have a fully assembled board.

You can download the Eagle files for the PCB below.

bleFeatherSynth.zip

https://adafru.it/IJF

The PCB was designed with the round tactile button switches in mind. They're also used for perfboard version and the 3D printed piano keys are sized for their round toppers.

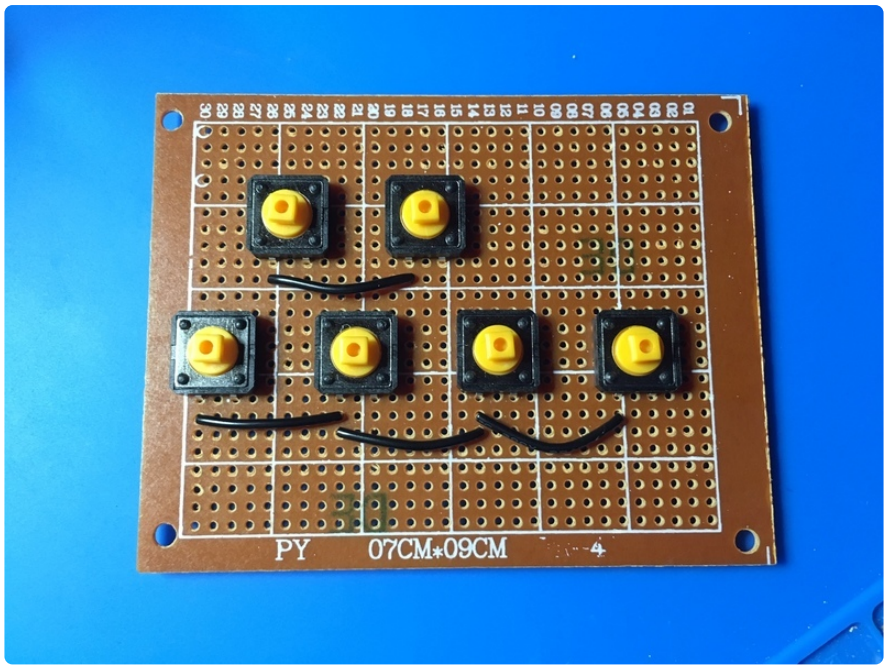
Colorful Round Tactile Button Switch Assortment - 15 pack
Little clicky switches are standard input "buttons" on electronic projects. These work best in a PCB but can be...
https://www.adafruit.com/product/1009

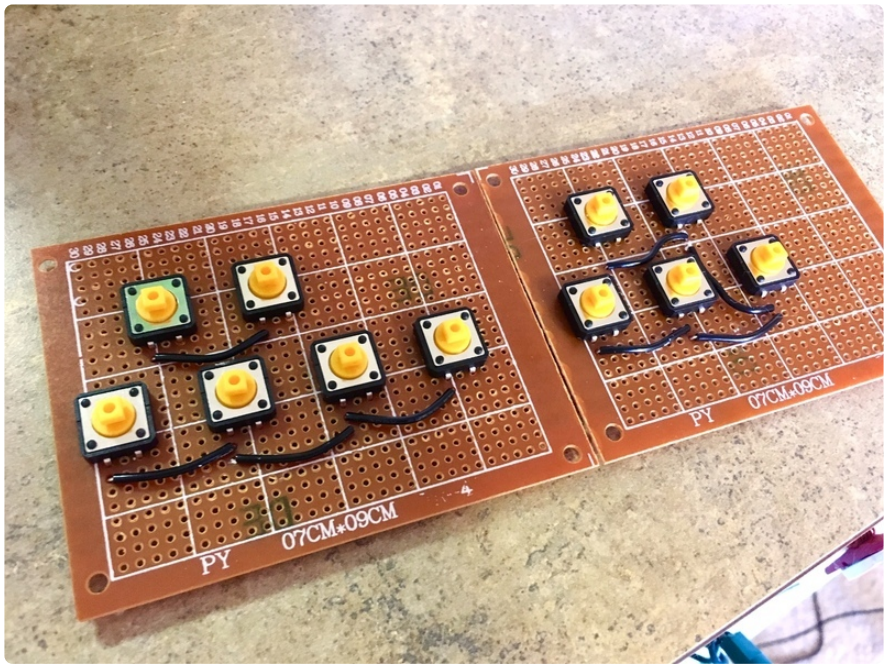
Perfboard

For the perfboard assembly, you need two pieces to start. Begin by placing 6 of the buttons on the first piece. The buttons should be placed 5 holes apart to have the spacing work for the 3D printed keys.

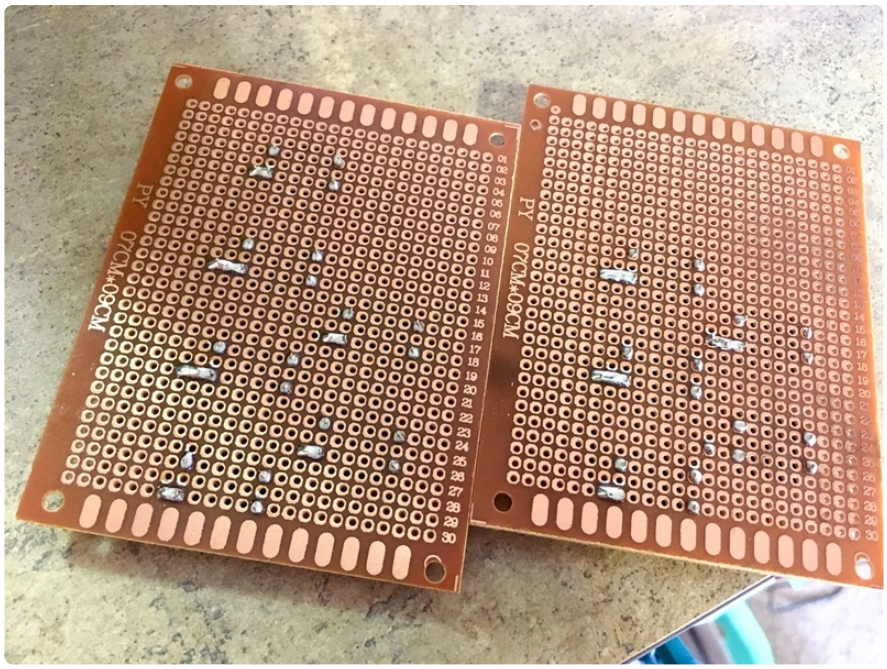
Cut pieces of wire for the **ground** and **input pins** for each button. The ground connections will daisy chain between each button, so they should be just long enough to reach the next button. The input wires will need to be long enough to reach where the **Feather** will eventually live on the far right side of the assembly.

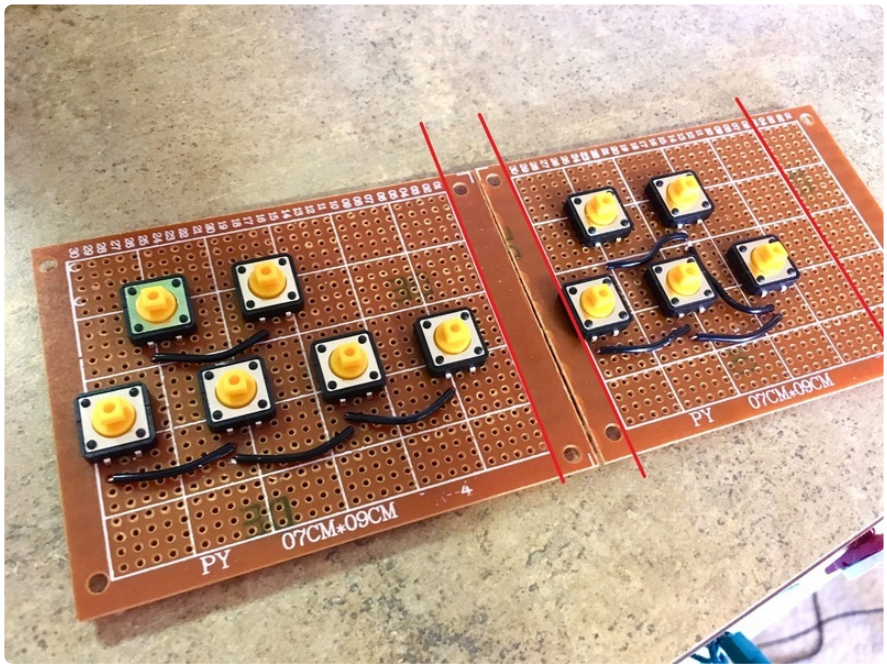
Repeat this process for the second piece of perfboard for five of the buttons.



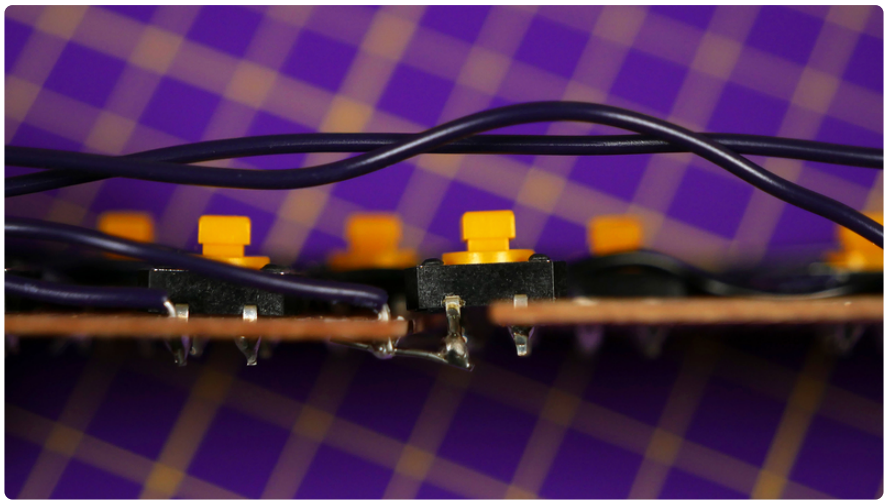


To make the connections on this type of perfboard, you'll need to bridge the individual solder points. For example, the input wire for the first button should have some solder that connects the two points.

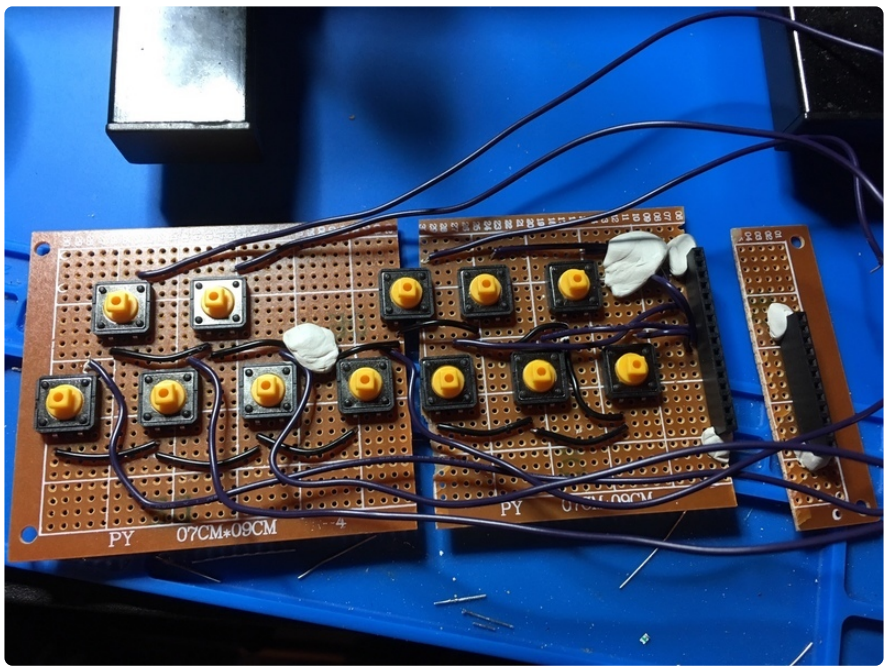




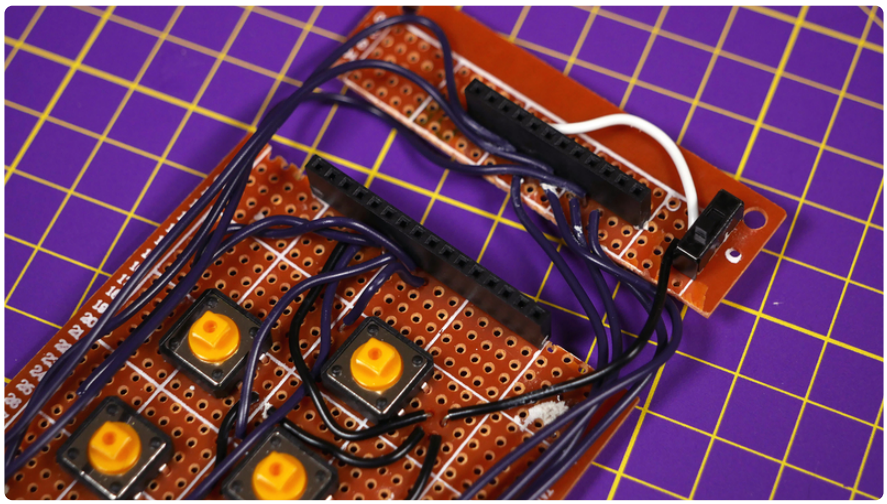
Next, cut the right edge off on the first piece and left edge off on the second piece. This will allow placement of a button, which will eventually be F#, in the middle. Place this button on the edge of the first piece, so that it's hanging off. Then run a short piece of wire to connect it to the second piece.



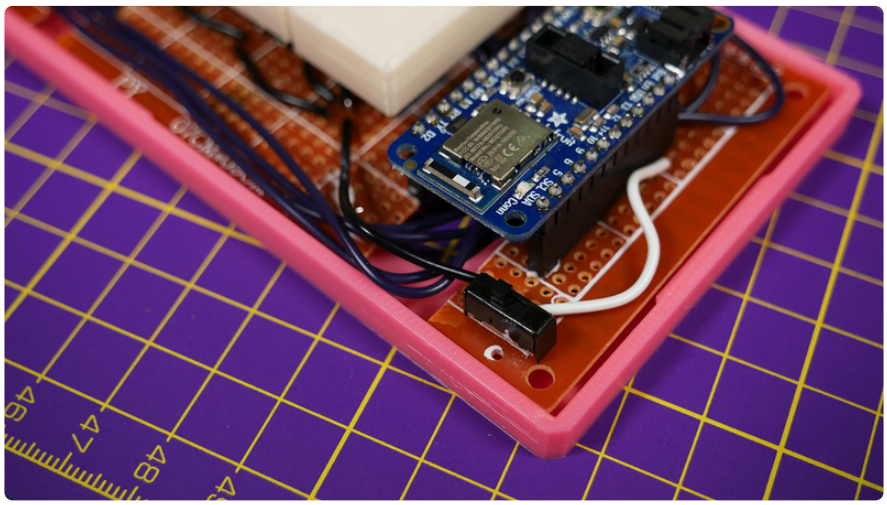
Then, we're going to cut another section: the last column of five rows on the right hand side of the second piece. Then solder on Feather headers as shown in the picture.



With the headers in place, connect the button wires to the Feather. Run the **input wires** to the appropriate **I/O pin** and then bridge the connections. Do the same with the **ground** signal.



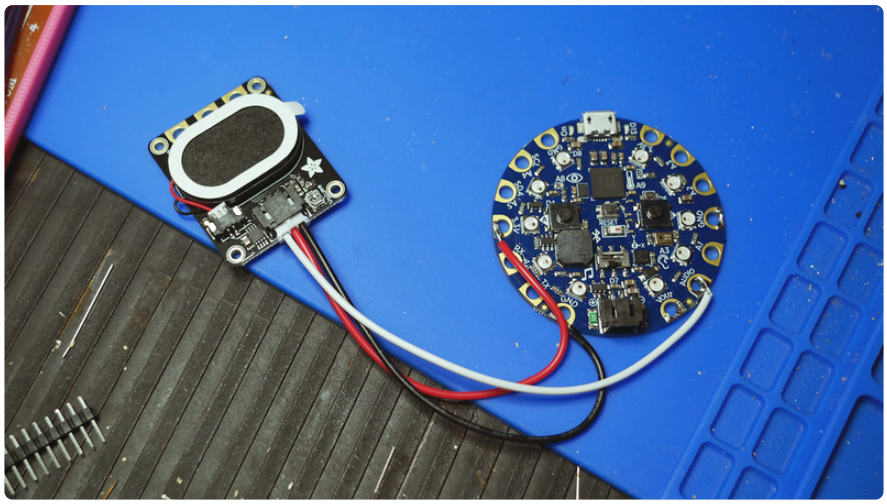
Finally, to add an **on/off switch**. Place it on the board and then run a wire from **GND** to one lead on the switch and then a wire from **EN** to the middle lead. Connecting **EN** to **GND** allows for built-in on/off control.



And there you have a perfboard equivalent to the PCB.

Soldering - Circuit Playground Bluefruit

To connect the **STEMMA Speaker**, use a STEMMA connector cable to plug into the STEMMA connector on the speaker's board. Then take those three wires and solder them to the Circuit Playground Bluefruit pins. **POWER** will be soldered to **3.3V**, **GND** will be soldered to **GND** and **SIGNAL** will be soldering to **AUDIO (A0)**.



CircuitPython Code Setup

Take your Feather nRF52840 and plug it into your computer via a known good data + power USB cable. Have your Circuit Playground Bluefruit handy as you'll be performing most of the same steps for each. Your operating system will show a drive named **CIRCUITPY** when a board is plugged in. If you get a drive named **CPLAYBTBOOT** or **FTHR840BOOT**, you'll likely need to install CircuitPython. This is easy, see the steps below to do this, get library files, and copy the code to each board.

Feather nRF52840 CircuitPython Setup

First, make sure you're running the most recent version of CircuitPython for the Feather nRF52840.

Click to read on installing CircuitPython on the Feather nRF52840

https://adafru.it/EwP

Circuit Playground Bluefruit Circuit Python Setup

Just like with the Feather, make sure you're running the most recent version of CircuitPython for the Circuit Playground Bluefruit.

Click to read on installing CircuitPython on the Circuit Playground Bluefruit

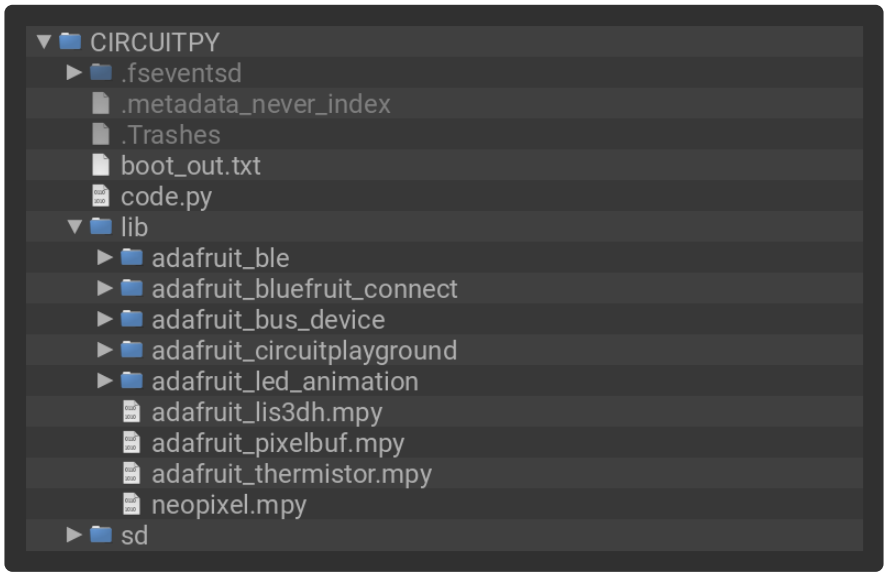
https://adafru.it/GYc

Download the Code from GitHub

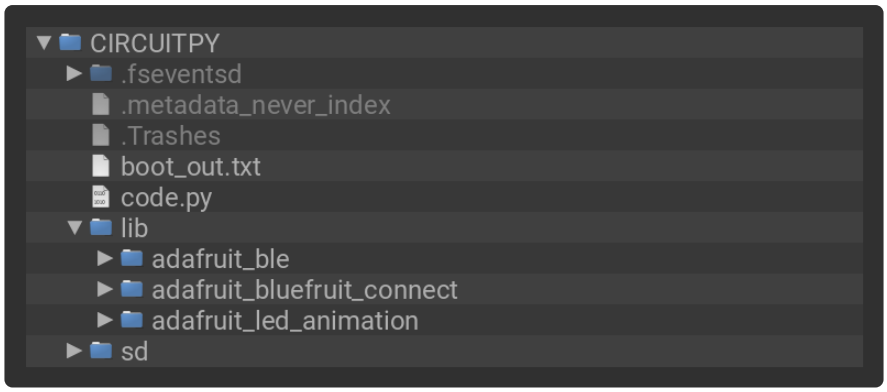
To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

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 **BLE_Synth/cpb_amp_code** for the Circuit Playground Bluefruit and **BLE_Synth/feather_keyboard_code** for the Feather nRF52840 and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your respective **CIRCUITPY** drives.

Your **CIRCUITPY** drive on the **Circuit Playground Bluefruit** should now look similar to the following image:



Your **CIRCUITPY** drive on the **Feather nRF52840** should now look similar to the following image:



Circuit Playground Bluefruit BLE Synth Code

```
# SPDX-FileCopyrightText: 2020 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

'''BLE Synth
File for the Circuit Playground Bluefruit
Amp Portion'''
from adafruit_circuitplayground.bluefruit import cpb
from adafruit_led_animation.animation import Comet, AnimationGroup,\
    AnimationSequence
import adafruit_led_animation.color as color
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService
from adafruit_bluefruit_connect.packet import Packet
from adafruit_bluefruit_connect.color_packet import ColorPacket
from adafruit_bluefruit_connect.button_packet import ButtonPacket

#  easily call for NeoPixels to be off
off = (0, 0, 0)
#  state to debounce on CPB end
tone = False

# Setup for comet animation
COMET_SPEED = 0.04  # Lower numbers increase the animation speed
CPB_COMET_TAIL_LENGTH = 5  # The length of the comet on the Circuit Playground
Bluefruit
CPB_COMET_BOUNCE = False  # Set to True to make the comet "bounce" the opposite
direction on CPB

animations = AnimationSequence(
    AnimationGroup(
        Comet(cpb.pixels, COMET_SPEED, off, tail_length=CPB_COMET_TAIL_LENGTH,
              bounce=CPB_COMET_BOUNCE)))

#  note frequencies
C4 = 261.63
Csharp4 = 277.18
D4 = 293.66
Dsharp4 = 311.13
E4 = 329.63
F4 = 349.23
Fsharp4 = 369.99
G4 = 392
Gsharp4 = 415.3
A4 = 440
Asharp4 = 466.16
B4 = 493.88
```

```
#  note array
note = [C4, Csharp4, D4, Dsharp4, E4, F4,
        Fsharp4, G4, Gsharp4, A4, Asharp4, B4]

#  colors to recieve from color packet & for neopixels
color_C = color.RED
color_Csharp = color.ORANGE
color_D = color.YELLOW
color_Dsharp = color.GREEN
color_E = color.TEAL
color_F = color.CYAN
color_Fsharp = color.BLUE
color_G = color.PURPLE
color_Gsharp = color.MAGENTA
color_A = color.GOLD
color_Asharp = color.PINK
color_B = color.WHITE

#  color array
color = [color_C, color_Csharp, color_D, color_Dsharp, color_E,
         color_F, color_Fsharp, color_G, color_Gsharp, color_A,
         color_Asharp, color_B]

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

while True:
    #  connect via BLE
    ble.start_advertising(advertisement)  # Start advertising.
    was_connected = False
    while not was_connected or ble.connected:
        if ble.connected:  # If BLE is connected...
            was_connected = True
            #  start animations
            animations.animate()
            #  look for packets
            if uart.in_waiting:
                try:
                    packet = Packet.from_stream(uart)  # Create the packet object.
                except ValueError:
                    continue
                #  if it's a color packet:
                if isinstance(packet, ColorPacket):
                    for i in range(12):
                        colors = color[i]
                        notes = note[i]
                        #  if the packet matches one of our colors:
                        if packet.color == colors and not tone:
                            #  animate with that color
                            animations.color = colors
                            #  play matching note
                            cpb.start_tone(notes)
                            tone = True
                #  if it's a button packet aka feather's button has been released:
                elif isinstance(packet, ButtonPacket) and packet.pressed:
                    if packet.button == ButtonPacket.RIGHT and tone:
                        tone = False
                        #  stop playing the note
                        cpb.stop_tone()
                        #  turn off the neopixels but keep animation active
                        animations.color = off
```

Feather nRF52840 BLE Synth Code

```
# SPDX-FileCopyrightText: 2020 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

'''BLE Synth
File for the Feather nFR52840
Keyboard Portion'''
import time
import board
import digitalio
import adafruit_led_animation.color as color
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService
from adafruit_bluefruit_connect.color_packet import ColorPacket
from adafruit_bluefruit_connect.button_packet import ButtonPacket

#  setup for LED to indicate BLE connection
blue_led = digitalio.DigitalInOut(board.BLUE_LED)
blue_led.direction = digitalio.Direction.OUTPUT

#  setting up the buttons
switch_pins = [board.D5, board.D6, board.D9, board.D10,
               board.D11, board.D12, board.D13, board.A0, board.A1, board.A2,
               board.A3, board.A4]
switch_array = []

#  creating the button array
for pin in switch_pins:
    switch_pin = digitalio.DigitalInOut(pin)
    switch_pin.direction = digitalio.Direction.INPUT
    switch_pin.pull = digitalio.Pull.UP
    switch_array.append(switch_pin)

#  states for button debouncing
switch1_pressed = False
switch2_pressed = False
switch3_pressed = False
switch4_pressed = False
switch5_pressed = False
switch6_pressed = False
switch7_pressed = False
switch8_pressed = False
switch9_pressed = False
switch10_pressed = False
switch11_pressed = False
switch12_pressed = False
switches_pressed = [switch1_pressed, switch2_pressed, switch3_pressed,
switch4_pressed,
                    switch5_pressed, switch6_pressed, switch7_pressed,
switch8_pressed,
                    switch9_pressed, switch10_pressed, switch11_pressed,
switch12_pressed]

#  colors from Animation library to send as color packets
#  named for notes
color_C = color.RED
color_Csharp = color.ORANGE
color_D = color.YELLOW
color_Dsharp = color.GREEN
color_E = color.TEAL
color_F = color.CYAN
color_Fsharp = color.BLUE
color_G = color.PURPLE
```

```
color_Gsharp = color.MAGENTA
color_A = color.GOLD
color_Asharp = color.PINK
color_B = color.WHITE

#  array for colors
color = [color_C, color_Csharp, color_D, color_Dsharp, color_E,
         color_F, color_Fsharp, color_G, color_Gsharp, color_A,
         color_Asharp, color_B]

#  BLE send_packet function
def send_packet(uart_connection_name, packet):
    """Returns False if no longer connected."""
    try:
        uart_connection_name[UARTService].write(packet.to_bytes())
    except:  # pylint: disable=bare-except
        try:
            uart_connection_name.disconnect()
        except:  # pylint: disable=bare-except
            pass
        return False
    return True

ble = BLERadio()

uart_connection = None

if ble.connected:
    for connection in ble.connections:
        if UARTService in connection:
            uart_connection = connection
        break

while True:
    blue_led.value = False
    #  BLE connection
    if not uart_connection or not uart_connection.connected:  # If not connected...
        print("Scanning...")
        for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):  #
Scan...
            if UARTService in adv.services:  # If UARTService found...
                print("Found a UARTService advertisement.")
                blue_led.value = True #  LED turns on when connected
                uart_connection = ble.connect(adv)  # Create a UART connection...
                break
        ble.stop_scan()  # And stop scanning.
    #  while connected..
    while uart_connection and uart_connection.connected:
    #  iterate through buttons and colors
        for switch_pin in switch_array:
            i = switch_array.index(switch_pin)
            switches_pressed_state = switches_pressed[i]
            colors = color[i]
            #  if the button is released
            #  worked best if placed before the button press portion
            if switch_pin.value and switches_pressed_state:
                print("button off")
                #  send button packet to stop tone & color (happens on CPB)
                if not send_packet(uart_connection,
                                   ButtonPacket(ButtonPacket.RIGHT, pressed=True)):
                    uart_connection = None
                    continue
                switches_pressed[i] = False  # Set to False.
                #  time delay for BLE, otherwise issues can arrise
                time.sleep(0.05)
            #  if button is pressed:
            if not switch_pin.value and not switches_pressed_state:  # If button A
pressed...
                #  send color packet
```

```
                if not send_packet(uart_connection, ColorPacket(colors)):
                    uart_connection = None
                    continue
                switches_pressed[i] = True  # Set to True.
                time.sleep(0.05)  # Debounce.
```

CircuitPython Code Walkthrough

Both the **Feather nrF52840** and **Circuit Playground Bluefruit** are running their own **code.py** file. In this case, the Feather's **code.py** file could be considered the A file and the Circuit Playground Bluefruit's **code.py** file could be considered the B file. The Feather is in charge because the Circuit Playground Bluefruit is waiting for the button inputs to be able to do anything.

Check out this Learn Guide for more info on BLE communication between boards

https://adafru.it/IKa

The Feather's code begins by setting up its **digital inputs** and **outputs**. The **on-board blue LED** will be used to indicate whether the boards have connected with each other via **BLE**. The rest of the **pins** are setup to be **button inputs** and are placed into the **array** `switch_array[]`.

```
blue_led = digitalio.DigitalInOut(board.BLUE_LED)
blue_led.direction = digitalio.Direction.OUTPUT

switch_pins = [board.D5, board.D6, board.D9, board.D10,
               board.D11, board.D12, board.D13, board.A0, board.A1, board.A2,
               board.A3, board.A4]
switch_array = []

for pin in switch_pins:
    switch_pin = digitalio.DigitalInOut(pin)
    switch_pin.direction = digitalio.Direction.INPUT
    switch_pin.pull = digitalio.Pull.UP
    switch_array.append(switch_pin)
```

Next, **state machines** are setup for the **button debouncing**. These states are then put into the array `switches_pressed[]`.

```
switch1_pressed = False
switch2_pressed = False
switch3_pressed = False
switch4_pressed = False
switch5_pressed = False
switch6_pressed = False
switch7_pressed = False
switch8_pressed = False
switch9_pressed = False
switch10_pressed = False
switch11_pressed = False
```

```
switch12_pressed = False
switches_pressed = [switch1_pressed, switch2_pressed, switch3_pressed,
switch4_pressed,
                    switch5_pressed, switch6_pressed, switch7_pressed,
switch8_pressed,
                    switch9_pressed, switch10_pressed, switch11_pressed,
switch12_pressed]
```

Going to the **Circuit Playground Bluefruit**, at the beginning of its **code.py** file, there is some setup for the **NeoPixels Animations library**. The **Comet** animation is being used to get a nice swirly effect each time a note is played.

```
COMET_SPEED = 0.04  # Lower numbers increase the animation speed
CPB_COMET_TAIL_LENGTH = 5  # The length of the comet on the Circuit Playground
Bluefruit
CPB_COMET_BOUNCE = False  # Set to True to make the comet "bounce" the opposite
direction on CPB

animations = AnimationSequence(
    AnimationGroup(
        Comet(cpb.pixels, COMET_SPEED, off, tail_length=CPB_COMET_TAIL_LENGTH,
        bounce=CPB_COMET_BOUNCE)))
```

Check out this Learn Guide on the Animation library

https://adafru.it/IKb

Speaking of notes, the tones that will be played through the **STEMMA Speaker** when the buttons are pressed are then setup. Variables matching the note and octave are set to match the correct **frequencies of the notes**. These are then put into an array called `note[]`.

```
C4 = 261.63
Csharp4 = 277.18
D4 = 293.66
Dsharp4 = 311.13
E4 = 329.63
F4 = 349.23
Fsharp4 = 369.99
G4 = 392
Gsharp4 = 415.3
A4 = 440
Asharp4 = 466.16
B4 = 493.88

note = [C4, Csharp4, D4, Dsharp4, E4, F4,
        Fsharp4, G4, Gsharp4, A4, Asharp4, B4]
```

This next portion has a bit of synergy between both **code.py** files. Previously it was mentioned that the Feather is sending **BLE packets** to the Circuit Playground Bluefruit, but what type? C**olor packets**. This means that the CPB has to know which colors to be listening for. As a result, both the Feather and CPB have identical colors defined. These are then put into an array in the same order. It so happens that both

files call this array `color[]`. The colors being used are also from the **Animations** library. They're predefined in that library so that we don't have to worry about dialing in the exact RGB values.

```
color_C = color.RED
color_Csharp = color.ORANGE
color_D = color.YELLOW
color_Dsharp = color.GREEN
color_E = color.TEAL
color_F = color.CYAN
color_Fsharp = color.BLUE
color_G = color.PURPLE
color_Gsharp = color.MAGENTA
color_A = color.GOLD
color_Asharp = color.PINK
color_B = color.WHITE

color = [color_C, color_Csharp, color_D, color_Dsharp, color_E,
            color_F, color_Fsharp, color_G, color_Gsharp, color_A,
            color_Asharp, color_B]
```

The next portions for both files are some **BLE** setup and the beginning of the **loop**. At the start of the loop, BLE connections are made between the two boards and if the connection is present then they can proceed to the synth portions of the loop.

```
#  CPB BLE portion

ble = BLERadio()
uart = UARTService()
advertisement = ProvideServicesAdvertisement(uart)

while True:
    ble.start_advertising(advertisement)  # Start advertising.
    was_connected = False
    while not was_connected or ble.connected:
        if ble.connected:  # If BLE is connected...
            was_connected = True
            animations.animate()

            if uart.in_waiting:  # Check to see if any data is available from the
Remote Control.
                try:
                    packet = Packet.from_stream(uart)  # Create the packet object.
                except ValueError:
                    continue
```

```
#  Feather BLE portion

def send_packet(uart_connection_name, packet):
    """Returns False if no longer connected."""
    try:
        uart_connection_name[UARTService].write(packet.to_bytes())
    except:  # pylint: disable=bare-except
        try:
            uart_connection_name.disconnect()
        except:  # pylint: disable=bare-except
            pass
        return False
    return True

ble = BLERadio()
```

```
uart_connection = None

if ble.connected:
    for connection in ble.connections:
        if UARTService in connection:
            uart_connection = connection
        break

while True:
    blue_led.value = False
    if not uart_connection or not uart_connection.connected:  # If not connected...
        print("Scanning...")
        for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):  #
Scan...
            if UARTService in adv.services:  # If UARTService found...
                print("Found a UARTService advertisement.")
                blue_led.value = True
                uart_connection = ble.connect(adv)  # Create a UART connection...
                break
        # Stop scanning whether or not we are connected.
        ble.stop_scan()  # And stop scanning.
```

For the Feather, there is a `for` statement that iterates through the `switch_array`, which holds the button pins. It defines the indexed position as `i`. Then `i` can be used to also track the debounce state and the **color** that is going to be sent as a color packet. All of these arrays have 12 indexes, one for each note that being used. This way, the indexes can easily be called out as needed from each array all at once without too many lines of code.

```
while uart_connection and uart_connection.connected:
        for switch_pin in switch_array:
            i = switch_array.index(switch_pin)
            switches_pressed_state = switches_pressed[i]
            colors = color[i]
```

Then there's a check to see if a button has been released with an `if` statement. You'll see that if it has, that a button packet is sent. You'll see why this is included when you see the CPB's loop in a moment.

```
if switch_pin.value and switches_pressed_state:  # On button release...
    print("button off")
    if not send_packet(uart_connection, #  ColorPacket(colors)):
                       ButtonPacket(ButtonPacket.RIGHT, pressed=True)):
        uart_connection = None
        continue
    switches_pressed[i] = False  # Set to False.
    time.sleep(0.05)
```

After that though, there's a check to see if a button has been pressed. If it has, then a color packet is sent. The color depends on the indexed color that matches the indexed button pin.

```
if not switch_pin.value and not switches_pressed_state:  # If button A pressed...
    if not send_packet(uart_connection, ColorPacket(colors)):
        uart_connection = None
```

```
            continue
        switches_pressed[i] = True  # Set to True.
        time.sleep(0.05)  # Debounce.
```

And that is how packets are being sent to the Circuit Playground Bluefruit. Now let's see what the Circuit Playground Bluefruit is doing when it receives those packets.

For the Circuit Playground Bluefruit, there is an **if** statement that checks if a color packet is being received. This is followed by a `for` statement that iterates through the `color` and `note` arrays.

```
if isinstance(packet, ColorPacket):  # If the packet is color packet...
    for i in range(12):
        colors = color[i]
        notes = note[i]
```

Then there's an **if** statement that checks to see if the **color packet** being sent by the Feather matches with any of the colors in our colors array. If it does, then a color is sent to the Comet animation and plays the matching tone. `tone` is also being used as a state machine to debounce our CPB.

```
if packet.color == colors and not tone:
    animations.color = colors
    cpb.start_tone(notes)
    tone = True
```

Following that, the loop ends with an `elif` statement. It's checking to see if a **`button packet`** is coming in. Remember, the Feather was setup to send a button packet once a physical button is released.

If the button packet is a `ButtonPacket.RIGHT` (which it will be), then the CPB will stop playing the tone. The animations will continue to animate but with the NeoPixels turned off.

`off` was defined earlier to equal `(0, 0, 0)`. If this portion of code wasn't included, then the last tone played would continue playing without stopping. The same would be the case for the NeoPixel animation.

```
elif isinstance(packet, ButtonPacket) and packet.pressed:  # If the packet is a
button packet...
    if packet.button == ButtonPacket.RIGHT and tone:  # If button B is pressed...
        tone = False
        cpb.stop_tone()
        animations.color = off
```

3D Printing

All of the 3D printing files can be downloaded directly from Thingiverse. Fusion360 files are also available for the individual models if you want to remix them.

Thngiverse Link for all of the .STL files

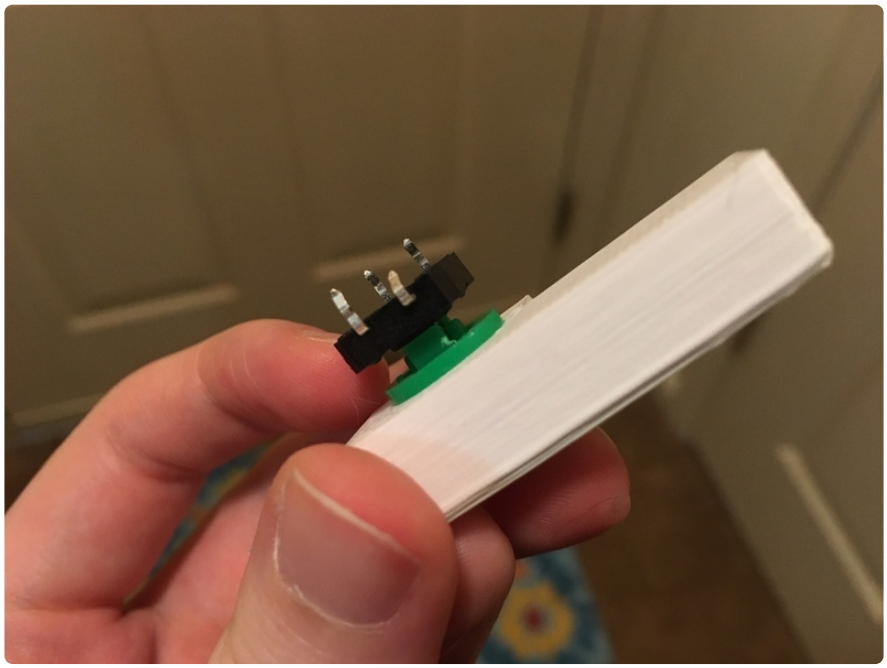
https://adafru.it/IKc

Keyboard Keys

To allow your Feather to live its best life as an 8-bit synth, you need to 3D print some keyboard keys. These keys are sized to snuggly fit over the circular top on these tactile buttons. They also print with no supports and are pretty cute.

Fusion360 File for the Keyboard Keys

https://adafru.it/IKd

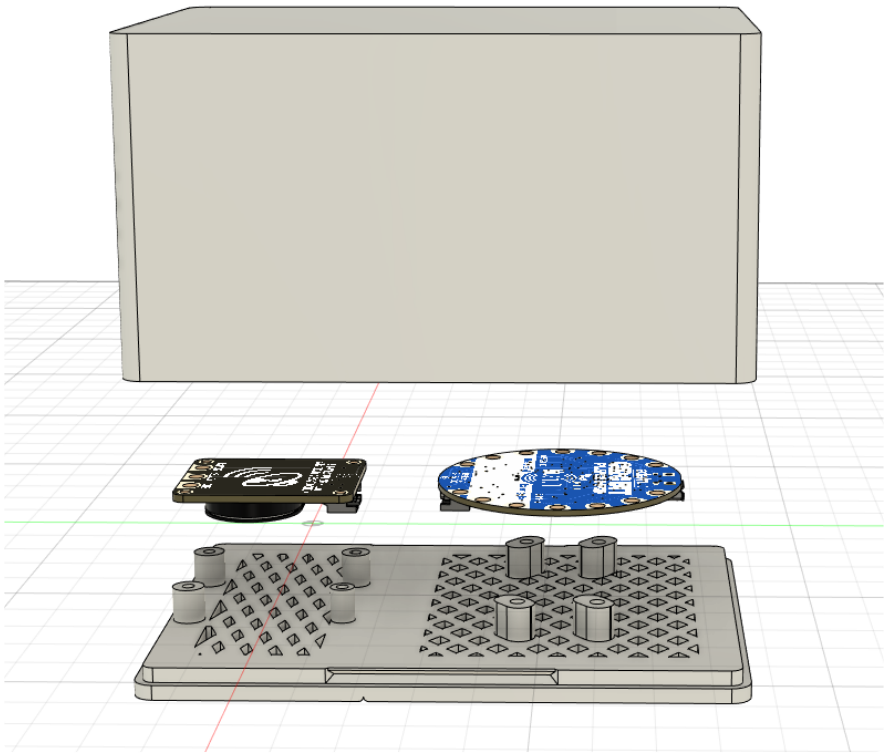


Circuit Playground Bluefruit Amp Enclosure

Since the **Circuit Playground Bluefruit** is acting as the amplifier for the synth, you might as well print a little amp-inspired case. The case is snap-fit and prints without supports. The front of the case has a pattern that make it look like an amp speaker grill. The **CPB** and **STEMMA Speaker** attach to the front via some **M3** and **M2.5** sized holes that allow for screws of the same size.

Fusion360 File for the Amp Case

https://adafru.it/IKe



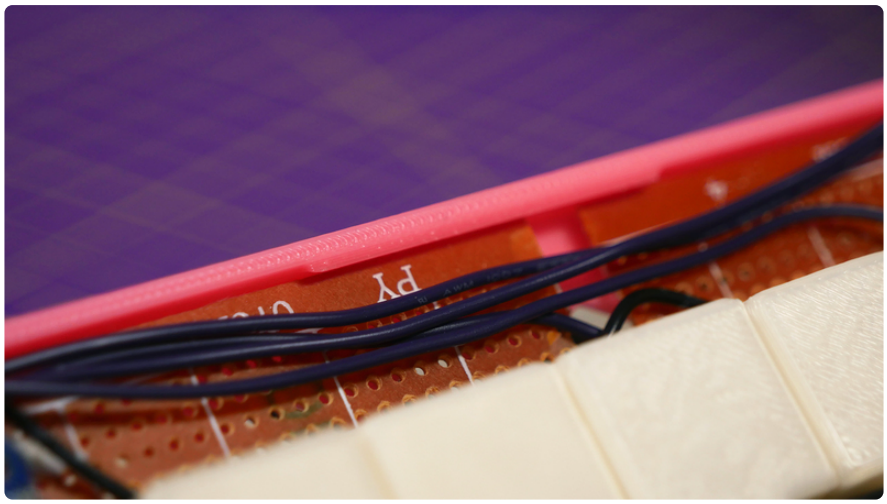
Feather Keyboard Enclosure Options

Depending on whether you're using the PCB or perfboard for the Feather portion of the circuit will determine your enclosure. 3D printed options are available for both, along with their Fusion360 files.

For the perfboard version, a shallow open box style enclosure is available with lips on the edges that keep it nice and secure.

Fusion360 File for the PerfBoard Keyboard Case

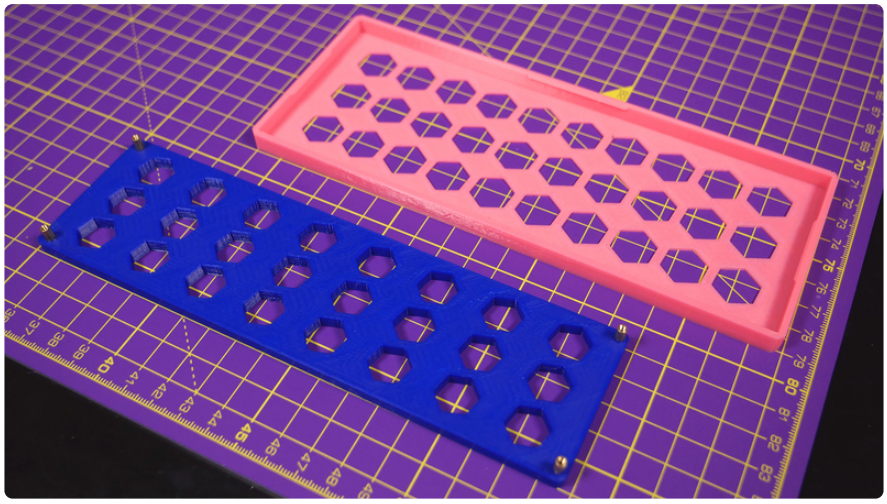
https://adafru.it/lKf



For the PCB, a base was created that allows for the PCB to sit on top with stand-offs. Much like the amp enclosure, it allows for screws or standoffs to be attached.

Fusion360 File for the PCB Keyboard Base

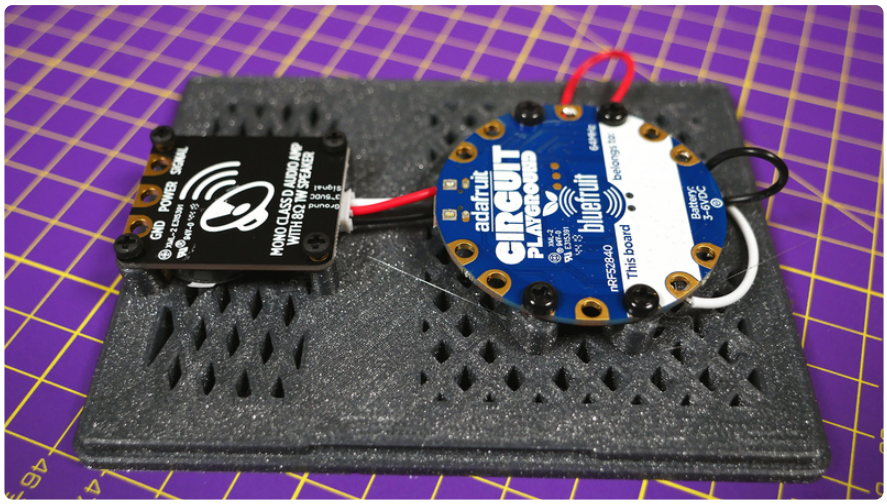
https://adafru.it/IKA



Assembly

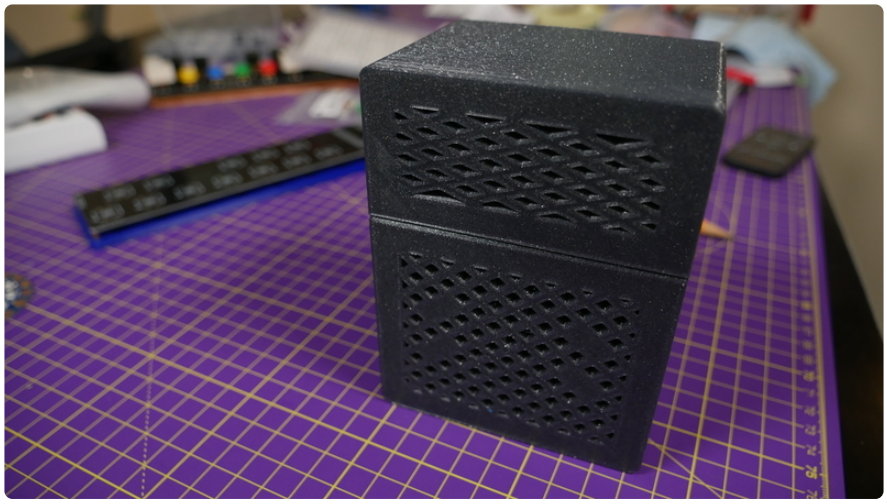
Circuit Playground Bluefruit Amp

On the interior of the lid of the amp case, there are mounting points for both the **Circuit Playground Bluefruit** and **STEMMA Speaker** that allow for some screws to attach the boards. The mounts for the Circuit Playground Bluefruit utilize **M3 sized screws** and the STEMMA Speaker utilizes **M2.5**.



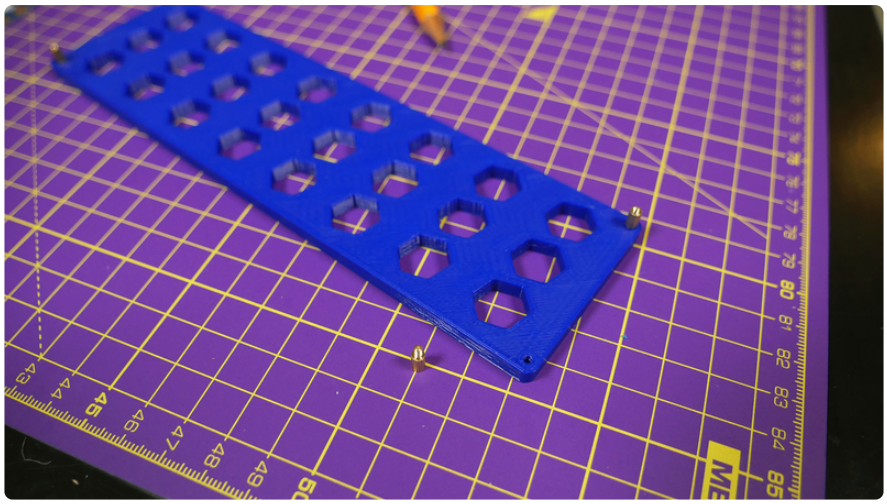
Using M3 screws, attach the Circuit Playground Bluefruit to the amp enclosure lid by lining up the holes with the front of the board facing down. Repeat the same process for the **STEMMA Speaker** with the front of the board again facing down.

Attach the **3x AAA battery pack** to the Circuit Playground Bluefruit's **JST battery connector** and then close up the amp. You're ready to amplify some 8-bit tones.



Feather Keyboard - PCB

The 3D printed base for the PCB allows for screws just like the **Circuit Playground Bluefruit**'s amp enclosure. Using some **M2 standoffs**, twist in the standoffs into the holes on the 3D printed base. Then place the PCB over the standoffs so that the mounting holes line up and attach with some M2 screws.



Feather Keyboard - Perfboard

The case for the perfboard version of the keyboard allows it to sit snuggly in the shallow box so that all of the electrical connections are protected and structural stability is provided. You can gently press the boards into the box and they should gently click in under the lips on the edges of the box.

Usage

Turn on both the **Circuit Playground Bluefruit** and **Feather nRF52840** keyboard. Once they're connected to each other, the **onboard blue LED** will turn on on the Feather. Then you should be able to press the keyboard buttons and hear your blippy tones blasting out of your CPB amp along with swirling NeoPixels.