Power Glove Wireless MIDI Controller
Created by John Park

https://learn.adafruit.com/power-glove-bluetooth-midi-controller

Last updated on 2023-08-29 04:25:44 PM EDT
# Table of Contents

## Overview
- Parts
- Parts
- Alternates

## CircuitPython on Feather Sense
- Set up CircuitPython Quick Start!

## Mod the Power Glove
- Input Methods
- Power Glove Disassembly
- Wrist Controller
- Palm Controller
- Palm PCB
- Wire Desoldering
- Sensor Testing/Mapping
- Power Glove to Feather Sense Circuit
- Mint Tins, Time Travel
- Solder the Reistors
- Solder the Sensor Leads
- Positive Voltage
- Wiring to the Wrist
- Ribbon Wiring
- Ribbon Connector
- Wiring Loom Connector
- Close the Wrist Controller
- Feather Wiring
- Power

## Code the Power Glove
- Setup
- Code
- Code Explainer
- Libraries
- Sense Setup
- Analog Read Pins
- MIDI CCs and Channel
- MIDI BLE Setup
- Get Flex CC Function
- Debug
- Wait for Connection
- Connected
- Flex
- Send MIDI Message
- Optional Message Types
- Disconnected

## Use the Power Glove MIDI Controller
Overview

Use a Nintendo Power Glove and Feather Sense running CircuitPython make a BLE MIDI synth gesture controller!

Use the four finger flex sensors in the Power Glove to send MIDI CC, Pitch Bend, or NoteOn/velocity info to your BLE MIDI synthesizer, including apps on iOS and computers. The Feather Sense nRF52840 has Bluetooth LE built in, as well as an on-board accelerometer used for tilt sensing.

No Power Glove? No problem -- build it with new flex sensors and your own glove.

Parts

You'll need the following parts for the build, including a Nintendo Power Glove, which you may have in your basement, or find at an auction online.
Parts

**Adafruit Feather nRF52840 Sense**
The Adafruit Feather Bluefruit Sense takes our popular Feather nRF52840 Express and adds a smorgasbord of sensors...
https://www.adafruit.com/product/4516

**Assembled Terminal Block Breakout FeatherWing for all Feathers**
The Terminal Block Breakout FeatherWing kit is like the Golden Eagle of prototyping FeatherWings (eg. majestic, powerful, good-looking). To start, you get a nice prototyping area...
https://www.adafruit.com/product/2926

**Lithium Ion Battery - 3.7V 2000mAh**
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...

If the Feather nRF52840 Sense is out of stock, advanced users should be able to adjust the code to use a regular Feather nRF52840 plus an acceleromter, such as the LSM6DOX, but this has not been tested so mileage may vary.
Silicone Cover Stranded-Core Ribbon Cable - 10 Wire 1 Meter Long
For those who are fans of our silicone-covered wires, but are always looking to up their wiring game. We now have Silicone Cover Ribbon cables! These may look...
[https://www.adafruit.com/product/3890](https://www.adafruit.com/product/3890)

Break-away 0.1" 36-pin strip male header - White - 10 pack
In this world, nothing can be said to be certain, except we need headers, headers, and more headers! Each pack contains ten white 36-pin 0.1"...
[https://www.adafruit.com/product/4149](https://www.adafruit.com/product/4149)

20-pin 0.1" Female Header - White - 5 pack
Female header is like the duct tape of electronics. It's great for connecting things together, soldering to perf-boards, sockets for wires or break-away header, etc. We go through...
[https://www.adafruit.com/product/4155](https://www.adafruit.com/product/4155)

Alternates

If you don't have a Power Glove, you can alternately build a BLE MIDI controller using any nRF52840 board and some flex sensors. These alternate boards also lack the accelerometer that's built onto the Feather Sense.
Long Flex sensor
This sensor can detect flexing or bending in one direction. They were popularized by being used in the Nintendo PowerGlove as a gaming interface. These sensors are easy to...
https://www.adafruit.com/product/182

Short Flex Sensor
This sensor can detect flexing or bending in one direction. They were popularized by being used in the Nintendo PowerGlove as a gaming interface. These sensors are easy to...
https://www.adafruit.com/product/1070

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

CircuitPython on Feather Sense

CircuitPython () is a derivative of MicroPython () designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the CIRCUITPY drive to iterate.
The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)
Plug your Feather Sense into your computer using a known-good 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 Reset button next to the USB connector on your board, and you will see the NeoPixel RGB LED turn green (identified by the arrow in the image). If it turns red, check the USB cable, try another USB port, etc. Note: The little red LED next to the USB connector will pulse red. That's ok!

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

You will see a new disk drive appear called FTHRSNSBOOT.

Drag the adafruit_circuitpython_etc.uf2 file to FTHRSNSBOOT.
The LED will flash. Then, the FTHRSNSBOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

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

Note: Some early release Sense boards had the drive named FTHR840BOOT. You can still copy .UF2s to the board, just copy to the board name appearing when the board is plugged in.

Mod the Power Glove
Input Methods
The stock Power Glove has three input methods: finger flex sensors, ultrasonic triangulation, and button pad entry.

Here you can see the thumb flex sensor (the glue had deteriorated on mine, so it was easy to peel up and have a look -- don't mess with yours if it's intact! I fixed mine up later with some E6000 glue.)

For our modified Power Glove MIDI controller, we'll use the four flex sensors and add tilt and roll sensing from the Feather Sense's accelerometer.
Power Glove Disassembly
Unscrew the four screws that hold the inner wrist support in place. Be sure to save the screws in a safe, organized place!

Unsnap the wrist band and remove it, then remove the inner wrist support and place them both aside.
Wrist Controller

The four screws you just removed not only held the wrist support in place, but also held the wrist control pad and main PCB in place.

Open the lid and set the base aside.

This is the main brain of the Power Glove.

Here you can see that the NES cable had already been desoldered and removed from mine. You can do the same now, we won't be needing it.
Palm Controller
The plastic casing on the back of the hand of the glove is sometimes referred to as the "palm controller" and "palm PCB". This contains circuitry and wiring for the flex sensors and the ultrasonic emitters, as well as a small piezo speaker.

A wiring loom runs a group of wires from the palm board to the wrist board.

Unscrew the four screws on the sides of the palm controller casing to open it up.

While the flex sensors themselves are capable of providing a smooth range of values when used in a voltage divider circuit, the Power Glove's designers limited their output to four positions each. 2-bits per flex sensor meant the four fingers' data fit into a single byte!
Palm PCB

A look at the palm PCB reveals some interesting things.

First of all, there's the type logo. Radofin made video game consoles (think Pong) and computers (ever heard of the Aquarius?) in the 1970s-80s, their involvement in the Power Glove doesn't get much mention. The Power Glove was actually made by Mattel on license to Nintendo, and Radofin designed electronics for Mattel, this makes sense.

We can see the four pairs of wires from the flex sensors are soldered to the PCB and then half of them run straight across the board to their wires that head through the wiring sleeve to the wrist PCB. The other half of the wires make a detour through a diode each first.

Some Power Glove modders have gone the route of desoldering the diodes and replacing them with resistors to create the voltage divider circuit we want, but out of respect to the board, I decided to desolder all of the wiring from it and remove the board. We'll then replace it with our own perma proto board.
Wire Desoldering

Using a hot soldering iron, some tweezers and proper ventilation, begin desoldering each of the flex sensor wires from the palm PCB.

If you have problems getting the solder to melt, it can help to add a bit of fresh flux core solder to the joint first.

Once the flex sensor wiring is off, desolder the wires from the other side of the palm PCB where they head to the wrist board.

Note, if you run into big blobs of gummy adhesive/hot glue obscuring the solder joints, you can remove it with isopropyl alcohol and tweezers.
Sensor Testing/Mapping
This is a great time to test out the original flex sensors, and determine which wire pair is connected to each finger. Using a multimeter in resistance mode, connect one end to a uniquely colored wires, such as white and then check each red wire until you measure a value around 160KΩ.

While still measuring these wires, bend the thumb and each of the first three fingers (sorry pinky) to determine which one is connected. The resistance value should smoothly increase to around 650KΩ as you bend the associated finger.

Map these out and you should end up with a chart similar to this:

- thumb = blue
- index = black
- middle = white
- ring = brown

We don't need to map each of the red wires since they'll all go to the positive 3.3V line.

Power Glove to Feather Sense Circuit
We'll create a voltage divider circuit for each flex sensor that allows us to read an analog value with the Feather for each sensor. This guide on using force sense resistors gives a great overview of how this will work.

With the 3.3V output of the Feather and a 470Ω fixed resistor to pull down to ground, we'll get about a 1V range out of each finger sensor.

Here's what the circuit will look like:
Mint Tins, Time Travel

Due to a complicated incident involving mint tins, neon scrunchies, and a time portal in Ladyada’s dorm room, the screw standoffs in the Power Glove palm molding match the mounting hole spacing of the Adafruit Perma-Proto Small Tin board.
Solder the Resistors
Grab your perma proto board and four 470KΩ resistors. Bend the leads in and insert them at an angle on the board as shown here. Note that one end of each resistor must go to the GND rail, indicated by the blue line on the board.

Bend the legs out to hold the resistors in place, then solder them to the board.

Clip the resistor leads off with flush cutters.
Solder the Sensor Leads

Now, solder the uniquely colored flex sensor wires each to their corresponding resistor.

Reference the Fritzing diagram above for the finger ordering, in case your colors don’t match exactly.

To keep things neat and tidy, you can insert these wires from the underside of the board and solder on top.
Positive Voltage

The other lead of each flex sensor will go to the positive 3.3V rail. Solder those wires (all red wires in the case of this particular unit) to the positive voltage rail, marked with the red line on the perma proto board.

Again, you can connect these from the bottom of the board and solder on the top side to keep things neat.
Wiring to the Wrist

We’ll reuse the wiring harness that we previously desoldered from the PCBs that connects the two sections of the glove.

Solder one wire per sensor row.

Match the wiring colors (or choose your own, so long as you keep track of them!) so we can make the proper connections to the Feather on the other end.

Solder the red and black wires to the positive voltage and GND rails, respectively.

You can reserve the unused wires for later use, just be sure to clip their exposed ends to avoid any shorts.
Ribbon Wiring

Next, run a 8" length of ribbon cable up from the wrist support area, through the small programming slit that is molded into the wrist controller base as shown here.

We'll only use six conductors, but I decided to keep all ten in case additional mods come up later, such as buttons wiring or adding LEDs.
Ribbon Connector

This idea came from Adafruit friend @todbot -- we’ll use header pin connectors to join the palm wiring to the ribbon cable.

Peel off the first six wires, using the white edged wire as the first position when counting.

Strip the ends and tin the wires.

Prepare a six position set of male and female headers as shown, marking one end as a reference with light colored paint marker.

Tin the female header connectors, then solder the wires in place as shown.
Wiring Loom Connector
Re-tin the ends of the six wires coming from the palm perma proto board through the wiring sleeve.

Insert the male header pin set into the female connector. Tin the exposed pins and solder the wiring in place as shown.

These are the positions we'll use:

Position 1 = red = voltage
Position 2 = black = GND
Position 3 = brown = ring
Position 4 = white = middle
Position 5 = gray = index
Position 6 = blue = thumb
Close the Wrist Controller

Add some Kapton tape and this connector at this end is complete.

Making sure to keep the wires out of the way of the mounting holes, you can now place the wrist pcb and cover/button pad on top of the wiring harness.
Feather Wiring

Now, we can flip the Power Glove over and wire up the Feather.

First, solder male headers to the Feather Sense and insert it into the Terminal Block FeatherWing.

Strip the ends of the six ribbon wires we'll use.

Then, insert the ribbon wiring into their respective positions on the screw terminal blocks as delineated in the wiring diagram above.
Power

Insert the LiPoly battery into the Feather's JST connector.

Use some double-sided adhesive foam tape or adhesive Velcro to secure the battery to the Power Glove wrist support space, and the FeatherWing to the battery.

You can still access the power switch on the FeatherWing and the USB port for programing and battery charging.
Your Power Glove MIDI controller is now wired up and ready for action.

---

**Code the Power Glove**

**Setup**

Set up the Feather Sense for use with CircuitPython by following [this guide](#).

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 PowerGlove_BLE_MIDI/ 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:
Here's how the code will work:

- On start, it will advertise itself as a Bluetooth Low Energy device named "Power Glove MIDI" with the BLE_MIDI service available, and wait for a connection.
- When a connection is made, it will read the Feather's accelerometer and the four flex sensors as analog inputs.
- These sensor readings are converted to values of 0-127
- The six sensor values are sent via BLE MIDI as continuous controller (CC) numbers 7, 70, 71, 75, and 77 over MIDI channel 1.
- This repeats continuously until the connection is broken or the Power Glove is powered off.

Code

Copy the code from the code-block below and paste it into the Mu editor and save it to your Feather as code.py (or copy code.py from the zip file and place on the CIRCUITPY drive).

```python
# SPDX-FileCopyrightText: 2021 John Park for Adafruit Industries.
# # SPDX-License-Identifier: MIT
#
# Power Glove BLE MIDI with Feather Sense nRF52840
# Sends MIDI CC values based on finger flex sensors and accelerometer
#
import time
import board
import adafruit_ble
doctor adafruit_ble.advertising.standard import ProvideServicesAdvertisement
import adafruit_ble_midi
import adafruit_midi
from adafruit_midi.control_change import ControlChange
# from adafruit_midi.note_on import NoteOn
# from adafruit_midi.pitch_bend import PitchBend
import adafruit_lsm6ds.lsm6ds33 # accelerometer
import simpleio
from analogio import AnalogIn
```
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller
sense_accel = adafruit_lsm6ds.lsm6ds33.LSM6DS33(i2c)

analog_in_thumb = AnalogIn(board.A3)
analog_in_index = AnalogIn(board.A2)
analog_in_middle = AnalogIn(board.A1)
analog_in_ring = AnalogIn(board.A0)

# Pick your MIDI CC numbers here
cc_x_num = 7  # volume
cc_y_num = 70  # unassigned
cc_thumb_num = 71  # unassigned
cc_index_num = 75  # unassigned
cc_middle_num = 76  # unassigned
cc_ring_num = 77  # unassigned

midi_channel = 1  # pick your midi out channel here

# Use default HID descriptor
midi_service = adafruit_ble_midi.MIDIService()
advertisement = ProvideServicesAdvertisement(midi_service)

ble = adafruit_ble.BLERadio()
if ble.connected:
    for c in ble.connections:
        c.disconnect()

midi = adafruit_midi.MIDI(midi_out=midi_service, out_channel=midi_channel - 1)

print("advertising")
ble.name="Power Glove MIDI"
ble.start_advertising(advertisement)

# reads an analog pin and returns value remapped to out range, e.g., 0-127
def get_flex_cc(sensor, low_in, high_in, min_out, max_out):
    flex_raw = sensor.value
    flex_cc = simpleio.map_range(flex_raw, low_in, high_in, min_out, max_out)
    flex_cc = int(flex_cc)
    return flex_cc

dbg = False  # set debug mode True to test raw values, set False to run BLE MIDI

while True:
    if dbg:
        accel_data = sense_accel.acceleration  # get accelerometer reading
        accel_x = accel_data[0]
        accel_y = accel_data[1]
        accel_z = accel_data[2]

        print("x: {} y: {} z: {} thumb: {} index: {} middle: {} ring: {}".format(accel_x, accel_y, accel_z, analog_in_thumb.value, analog_in_index.value, analog_in_middle.value, analog_in_ring.value,
        )
    time.sleep(0.2)

    else:
        print("Waiting for connection")
        while not ble.connected:
pass

print("Connected")

while ble.connected:
    # Feather Sense accelerometer readings to CC
    accel_data = sense.acceleration  # get accelerometer reading
    accel_x = accel_data[0]
    accel_y = accel_data[1]
    # accel_z = accel_data[2]
    # Remap analog readings to cc range
    cc_x = int(simpleio.map_range(accel_x, 0, 9, 127, 0))
    cc_y = int(simpleio.map_range(accel_y, 1, -9, 0, 127))
    cc_thumb = get_flex_cc(analog_in_thumb, 49000, 35000, 127, 0)
    cc_index = get_flex_cc(analog_in_index, 50000, 35000, 0, 127)
    cc_middle = get_flex_cc(analog_in_middle, 55000, 40000, 0, 127)
    cc_ring = get_flex_cc(analog_in_ring, 55000, 42000, 0, 127)
    
    print("CC_X:{} CC_Y:{} CC_Thumb:{} CC_Index:{} CC_Middle:{} CC_Ring:").format(
        cc_x, cc_y, cc_thumb, cc_index, cc_middle, cc_ring
    )
    
    # send all the midi messages in a list
    midi.send(
        [ControlChange(cc_x_num, cc_x),
         ControlChange(cc_y_num, cc_y),
         ControlChange(cc_thumb_num, cc_thumb),
         ControlChange(cc_index_num, cc_index),
         ControlChange(cc_middle_num, cc_middle),
         ControlChange(cc_ring_num, cc_ring),
        ]
    )
    
    # If you want to send NoteOn or Pitch Bend, here are examples:
    # midi.send(NoteOn(44, 120))  # G sharp 2nd octave
    # a_pitch_bend = PitchBend(random.randint(0, 16383))
    # midi.send(a_pitch_bend)

print("Disconnected")

print()

ble.start_advertising(advertisement)

---

**Code Explainer**

Here’s how the code works, and some areas that are ready for customization.

**Libraries**

First we’ll import the necessary libraries. These include `time`, `board`, `adafruit_lsm6ds`, `simpleio`, and `analogio` to deal with pauses, board hardware access, LSM6DS accelerometer reading, IO, and analog reads.

We also import `adafruit_ble`, `adafruit_ble_midi`, and `adafruit_midi` to enable Bluetooth LE control, sending BLE MIDI messages, and general MIDI commands.
import time
import board
import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
import adafruit_ble_midi
import adafruit_midi
from adafruit_midi.control_change import ControlChange
# from adafruit_midi.note_on import NoteOn
# from adafruit_midi.pitch_bend import PitchBend
import adafruit_lsm6ds # accelerometer
import simpleio
from analogio import AnalogIn

You can uncomment the NoteOn and PitchBend imports if you choose to send those kinds of MIDI messages.

Sense Setup

The Feather Sense's accelerometer is on the I2C bus, so we'll set that up next.

```python
i2c = board.I2C()
sense_accel = adafruit_lsm6ds.LSM6DS33(i2c)
```

Analog Read Pins

The finger flex sensors will be read on four analog input pins -- we'll set them up with nice names.

```python
analog_in_thumb = AnalogIn(board.A3)
analog_in_index = AnalogIn(board.A2)
analog_in_middle = AnalogIn(board.A1)
analog_in_ring = AnalogIn(board.A0)
```

MIDI CCs and Channel

We'll create variables for each of the six sensors we'll be reading to assign them to MIDI CC numbers.

Some CC numbers have standardized assignments on most synthesizers and synth software, while others are wide open to your own assignments. Here's a good list to consult(). I chose:

- For Power Glove tilt I chose CC 7, which is typically used for overall volume.
- For Power Glove roll, CC 70, which is a vague "sound variation" control, but you should be able to assign as you like.
• Thumb flex is CC 71, typically filter resonance.
• Index flex is CC 75, generic, so you can assign freely.
• Middle flex is CC 76, generic.
• Ring flex is CC 77, generic.

We'll also set the MIDI output to channel 1. You can pick any channel from 1-16 depending on your needs.

```
cx_num = 7  # volume
cy_num = 70  # unassigned
cthumb_num = 71  # unassigned
cindex_num = 75  # unassigned
cmiddle_num = 76  # unassigned
cring_num = 77  # unassigned

midi_channel = 1  # pick your midi out channel here
```

MIDI BLE Setup

Now we will set up the BLE MIDI service, and set the BLE radio to advertise the device, with the name "Power Glove MIDI".

```
midi_service = adafruit_ble_midi.MIDIService()
advertising = ProvideServicesAdvertisement(midi_service)

ble = adafruit_ble.BLERadio()
if ble.connected:
    for c in ble.connections:
        c.disconnect()

midi = adafruit_midi.MIDI(midi_out=midi_service, out_channel=midi_channel - 1)

print("advertising")
ble.name="Power Glove MIDI"
ble.start_advertising(advertisement)
```

Get Flex CC Function

To enable reading of the flex sensors and converting their values from a 16-bit analog read to a 7-bit MIDI CC value, we'll create a function called `get_flex_cc()`. This function takes five arguments: sensor name to read, low input value, high input value, minimum output value, and maximum output value.

When it is called it will return a CC value somewhere between 0-127.

```
def get_flex_cc(sensor, low_in, high_in, min_out, max_out):
    flex_raw = sensor.value
    flex_cc = simpleio.map_range(flex_raw, low_in, high_in, min_out, max_out)
```
flex_cc = int(flex_cc)
return flex_cc

**Debug**

In order to facilitate a type of manual sensor calibration you can turn the `debug` variable from `False` to `True`.

In the main loop of the program, when `debug` is `True` the raw accelerometer and flex sensor data will print to the serial port (REPL in Mu). This way you can check the value ranges and adjust the ranges in the main code to match.

```python
def sense_data():
    return

def sense_acceleration():
    return

def sense_accelerometer():
    return

while True:
    if debug:
        accel_data = sense_accelerometer() # get accelerometer reading
        accel_x = accel_data[0]
        accel_y = accel_data[1]
        accel_z = accel_data[2]

        print(
            "x:{}, y:{}, z:{}, thumb:{}, index:{}, middle:{}, ring:{}".format(  
                accel_x,
                accel_y,
                accel_z,
                analog_in_thumb.value,
                analog_in_index.value,
                analog_in_middle.value,
                analog_in_ring.value,
            )
        )

        time.sleep(0.2)
```

**Wait for Connection**

When we're out of debug mode, the first thing that happens in the main loop is that the BLE connection will be tested. It will remain in this state until a connection is made.

```python
print("Waiting for connection")
while not ble.connected:
    pass
print("Connected")
```

**Connected**

Once the Feather has connected to a BLE MIDI device, we proceed with checking all of the sensors.
First, the accelerometer values are read, and then the \(x\) (tilt) and \(y\) (roll) values are remapped as integers between 0-127 and cast to a pair of variables named \(cc_x\) and \(cc_y\).

You can see here that the \(cc_x\) is set to send a value of 127 when the accelerometer x-axis reads 0, meaning the volume will be full when the Power Glove is parallel to the ground. When you tilt the Power Glove up toward the sky, the accelerometer x-axis will read around 9, and the CC value will be set to 0, so the volume will be turn down to nothing.

```python
accel_data = sense_accel.acceleration  # get accelerometer reading
accel_x = accel_data[0]
accel_y = accel_data[1]
# accel_z = accel_data[2]
# Remap analog readings to CC range
cc_x = int(simpleio.map_range(accel_x, 0, 9, 127, 0))
cc_y = int(simpleio.map_range(accel_y, 1, -9, 0, 127))
```

If you wanted to reverse the tilt/volume behavior you could specify the remapping as:

```python
cc_x = int(simpleio.map_range(accel_x, 0, 9, 0, 127))
```

**Flex**

Then, the four variables for the flex sensors are cast with the remapped values returned by the `get_flex_cc()` function.

Note the values for the upper and lower ranges were determined first by running the program in debug mode and writing down those values.

Again, you can swap the "knob direction" of any of these by swapping the min-max CC values called from 0-127 or 127-0.

```python
cc_thumb = get_flex_cc(analog_in_thumb, 49000, 35000, 127, 0)
cc_index = get_flex_cc(analog_in_index, 50000, 35000, 0, 127)
cc_middle = get_flex_cc(analog_in_middle, 55000, 40000, 0, 127)
cc_ring = get_flex_cc(analog_in_ring, 55000, 42000, 0, 127)
```

**Send MIDI Message**

With all of the CC values read, we'll now send a series of 8-bit MIDI messages. You can call the `midi.send()` command individually for each MIDI message, or call the command as a list. (They'll all still be sent as individual MIDI messages.)

```python
midi.send([    ]
```
These will be transmitted in near real-time to the MIDI device, but not instantaneously -- BLE MIDI latency runs between 7.5 ms-22 ms typically. This isn't really noticeable with most CC parameters or playing notes with soft attacks, but could be bothersome with percussive, sharp attack NoteOn messages.

Optional Message Types

If you uncomment the necessary library imports, you can try out the NoteOn and PitchBend message types.

These lines of commented code show the essentials.

```python
# midi.send(NoteOn(44, 120))  # G sharp 2nd octave
# a_pitch_bend = PitchBend(random.randint(0, 16383))
# midi.send(a_pitch_bend)
```

Disconnected

If the connection is broken, the `ble.start_advertising(advertisement)` will begin again.

```python
print("Disconnected")
print()
ble.start_advertising(advertisement)
```

Use the Power Glove MIDI Controller

The video above shows the Power Glove MIDI controller in action. In it, the Power Glove has been paired with a mac os computer with the MIDI CC values have been mapped to various controls in the software modular synthesizer [VCV Rack](https://www.vcvrack.com/).

To use the Power Glove MIDI controller with a mac os machine, follow these setup instructions.
On iOS devices, you can use the MIDI Wrench app () to create and test the BLE MIDI connection, and then use it to "port" the MIDI messages to other apps, such as Audio Kit Synth One () and Moog Minimoog () which you can see in action below:

Most of these software synths will provide a way to map CC values to any function you like, often with a "MIDI learn" mode as well as explicit number entry.

You can also use BLE MIDI with some hardware synthesizers, as well as with BLE MIDI add-on dongles, such as this one from Yamaha () which adds BLE MIDI to synths with classic DIN-5 MIDI.

Enjoy your Power Glove wireless MIDI controller. It's so bad!