4x4 Rotary Encoder MIDI Messenger

Created by Liz Clark

https://learn.adafruit.com/4x4-rotary-encoder-midi-messenger

Last updated on 2023-11-13 10:17:28 AM EST
# Table of Contents

## Overview
- Prerequisite Guides
- Parts

## Circuit Diagram

## 3D Printing
- Optional: Rotary Encoder Riser

## Install CircuitPython
- CircuitPython Quickstart
- Safe Mode
- Flash Resetting UF2

## Code the MIDI Messenger
- Upload the Code and Libraries to the Feather RP2040
- How the CircuitPython Code Works
- Display
- seesaw Setup
- Interrupts
- Encoders, Switches and More Interrupts
- NeoPixels
- Tracking Encoder Positions
- Classes
- Tasks
- Run!

## Soldering
- Headers
- Encoders
- I2C Address Jumpers
- Interrupt Pins
- STEMMA QT Cables

## Assembly
- Mount the FeatherWing Doubler
- Mount the OLED
- Mount the Rotary Encoders
- Knobs

## Use
Overview

In this project, you'll build a MIDI controller that uses 16 rotary encoders soldered to quad rotary encoder breakouts that communicate over I2C. Usually this would mean continuously polling the I2C port to check for any changes in status, which can result in a slow response time. However, this CircuitPython code uses `asyncio` and the interrupt pins on the breakouts to make the response nice and speedy for sending your MIDI control change messages.

This MIDI controller is designed to send MIDI control change (CC) messages to other MIDI hardware, such as guitar pedals or hardware synthesizers. You can dial in the value that you want to send with the encoder and then press the switch on the encoder to send the CC message.
A small OLED display shows the CC value and message number queued up to send. When a message is sent, you'll see "Sent!" text appear in the bottom right-hand corner.

As you turn the rotary encoders, the NeoPixels underneath the encoder will cycle through the colors of the rainbow. The onboard NeoPixel on the Feather RP2040 will also advance through the colors. This is able to happen without blocking any of the MIDI functionality thanks to `asyncio`.

Prerequisite Guides

- Adafruit MIDI FeatherWing
- Cooperative Multitasking in CircuitPython with asyncio
Parts

Adafruit Feather RP2040
A new chip means a new Feather, and the Raspberry Pi RP2040 is no exception. When we saw this chip we thought "this chip is going to be awesome when we give it the Feather..."
https://www.adafruit.com/product/4884

Adafruit MIDI FeatherWing Kit
Turn your Feather into a song-bird with this musically-enabled FeatherWing that adds MIDI input and output jacks to just about any Feather. You get both input and output DIN-5 MIDI...
https://www.adafruit.com/product/4740

Adafruit I2C Quad Rotary Encoder Breakout with NeoPixel
Rotary encoders are soooo much fun! Twist em this way, then twist them that way. Unlike potentiometers, they go all the way around and often have little detents for tactile feedback....
https://www.adafruit.com/product/5752
5 x STEMMA QT cable
100 mm

https://www.adafruit.com/product/4210

4 x Clear Micro Potentiometer Knob
4 pack

https://www.adafruit.com/product/5676

1 x Pink and Purple Woven USB A to USB C Cable
1 meter long

https://www.adafruit.com/product/5153

Rotary Encoder + Extras
This rotary encoder is the best of the best, it's a high-quality 24-pulse encoder, with detents and a nice feel. It is panel mountable for placement in a box, or you can plug it...
https://www.adafruit.com/product/377

Monochrome 0.96" 128x64 OLED Graphic Display - STEMMA QT
These displays are small, only about 1" diagonal, but very readable due to the high contrast of an OLED display. This display is made of 128x64 individual white OLED pixels, each...
https://www.adafruit.com/product/326

FeatherWing Doubler - Prototyping Add-on For All Feather Boards
This is the FeatherWing Doubler - a prototyping add-on and more for all Feather boards. This is similar to our...
https://www.adafruit.com/product/2890
The OLED screen and four quad rotary encoder breakouts are daisy chained together with STEMMA QT cables. These boards are plugged into the STEMMA QT port on the Feather RP2040.

The Fritzing diagram has the I2C addresses for each quad rotary encoder breakout listed and the Soldering page in this guide walks through which jumpers need to be adjusted for each breakout.

The Feather and MIDI FeatherWing are plugged into a FeatherWing Doubler.

The code uses interrupts from the quad rotary encoder breakouts to determine if a button has been pressed or if a rotary encoder has been moved. The INT pins on the breakouts are soldered to digital inputs on the Feather RP2040.

- Breakout 0x49 INT to Feather D5 (green wire)
• Breakout 0x4A INT to Feather D6 (grey wire)
• Breakout 0x4B INT to Feather D9 (white wire)
• Breakout 0x4C INT to Feather D10 (orange wire)

3D Printing

The build may be housed in a 3D printed enclosure described below. It consists of two parts: the case and the lid. The lid attaches to the case with M3 screws.

The lid has cutouts and mounting holes for the rotary encoders and the OLED screen.

The case has cutouts for the USB C port and MIDI ports. There are mounting holes and support columns for the rotary encoder breakouts.
Optional: Rotary Encoder Riser

For this build, the NeoPixels light up in a rainbow pattern, but do not provide any feedback for the MIDI messages. If you want to see the pretty lights in your build though, you can print these risers for the rotary encoders to sit on. The risers are just tall enough to diffuse the NeoPixels while still letting the rotary encoder pins be soldered into the breakout PCB.

Install CircuitPython

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.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

Download the latest version of CircuitPython for this board via circuitpython.org
Click the link above to download the latest CircuitPython UF2 file. Save it wherever is convenient for you.

To enter the bootloader, hold down the BOOT/BOOTSEL button (highlighted in red above), and while continuing to hold it (don't let go!), press and release the reset button (highlighted in blue above). Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!

If the drive does not appear, release all the buttons, and then repeat the process above.

You can also start with your board unplugged from USB, press and hold the BOOTSEL button (highlighted in red above), continue to hold it while plugging it into USB, and wait for the drive to appear before releasing the button.

A lot of people end up using charge-only USB cables and it is very frustrating! Make sure you have a USB cable you know is good for data sync.
You will see a new disk drive appear called RPI-RP2.

Drag the adafruit_circuitpython_etc.uf2 file to RPI-RP2.

The RPI-RP2 drive will disappear and a new disk drive called CIRCUITPY will appear.

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

Safe Mode

You want to edit your code.py or modify the files on your CIRCUITPY drive, but find that you can't. Perhaps your board has gotten into a state where CIRCUITPY is read-only. You may have turned off the CIRCUITPY drive altogether. Whatever the reason, safe mode can help.
Safe mode in CircuitPython does not run any user code on startup, and disables auto-reload. This means a few things. First, safe mode bypasses any code in boot.py (where you can set CIRCUITPY read-only or turn it off completely). Second, it does not run the code in code.py. And finally, it does not automatically soft-reload when data is written to the CIRCUITPY drive.

Therefore, whatever you may have done to put your board in a non-interactive state, safe mode gives you the opportunity to correct it without losing all of the data on the CIRCUITPY drive.

Entering Safe Mode

To enter safe mode when using CircuitPython, plug in your board or hit reset (highlighted in red above). Immediately after the board starts up or resets, it waits 1000ms. On some boards, the onboard status LED (highlighted in green above) will blink yellow during that time. If you press reset during that 1000ms, the board will start up in safe mode. It can be difficult to react to the yellow LED, so you may want to think of it simply as a slow double click of the reset button. (Remember, a fast double click of reset enters the bootloader.)

In Safe Mode

If you successfully enter safe mode on CircuitPython, the LED will intermittently blink yellow three times.

If you connect to the serial console, you'll find the following message.

```
Auto-reload is off.
Running in safe mode! Not running saved code.
CircuitPython is in safe mode because you pressed the reset button during boot.
Press again to exit safe mode.
Press any key to enter the REPL. Use CTRL-D to reload.
```

You can now edit the contents of the CIRCUITPY drive. Remember, your code will not run until you press the reset button, or unplug and plug in your board, to get out of safe mode.

Flash Resetting UF2

If your board ever gets into a really weird state and doesn't even show up as a disk drive when installing CircuitPython, try loading this 'nuke' UF2 which will do a 'deep
clean' on your Flash Memory. You will lose all the files on the board, but at least you'll be able to revive it! After loading this UF2, follow the steps above to re-install CircuitPython.

Download flash erasing "nuke" UF2

Code the MIDI Messenger

Once you've finished setting up your Feather RP2040 with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the Download Project Bundle button in the window below. It will download as a zipped folder.

```python
# SPDX-FileCopyrightText: 2023 Liz Clark for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import asyncio
import board
import digitalio
from rainbowio import colorwheel
import keypad
import displayio
import busio
import adafruit_seesaw.seesaw
import adafruit_seesaw.neopixel
import adafruit_seesaw.rotaryio
import adafruit_seesaw.digitalio
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text import label
import adafruit_displayio_ssd1306
import adafruit_midi
from adafruit_midi.control_change import ControlChange
import neopixel

# default MIDI channel (1-16)
midi_in_channel = 2
midi_out_channel = 2

# MIDI CC messages, values and names assigned to each encoder
cc_values = [
    {'cc_val': (0, 127), 'cc_message': (14), 'cc_name': "Volume"},
    {'cc_val': (0, 127), 'cc_message': (15), 'cc_name': "Repeats"},
    {'cc_val': (0, 127), 'cc_message': (16), 'cc_name': "Size"},
    {'cc_val': (0, 127), 'cc_message': (17), 'cc_name': "Mod"},
    {'cc_val': (0, 127), 'cc_message': (18), 'cc_name': "Spread"},
    {'cc_val': (0, 127), 'cc_message': (19), 'cc_name': "Scan"},
    {'cc_val': (0, 127), 'cc_message': (20), 'cc_name': "Ramp"},
    {'cc_val': (1, 3), 'cc_message': (21), 'cc_name': "Mod Number"},
    {'cc_val': (1, 3), 'cc_message': (22), 'cc_name': "Mode"},
    {'cc_val': (0, 1), 'cc_message': (23), 'cc_name': "Bypass/Engage"},
    {'cc_val': (0, 127), 'cc_message': (24), 'cc_name': "Loop (R Hold)"},
    {'cc_val': (0, 1), 'cc_message': (25), 'cc_name': "Scan (L Hold)"},
    {'cc_val': (0, 127), 'cc_message': (26), 'cc_name': "Clear (Both Hold)"},
    {'cc_val': (0, 1), 'cc_message': (51), 'cc_name': "MIDI Clock Ignore"}
]```
displayio.release_displays()

oled_reset = board.D13

i2c = board.STEMMA_I2C()

# STEMMA OLED setup
display_bus = displayio.I2CDisplay(i2c, device_address=0x3D, reset=oled_reset)
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=128, height=64)

splash = displayio.Group()
display.root_group = splash
font = bitmap_font.load_font('/OCRA_small.pcf')

# main label/MIDI message name text; centered
main_area = label.Label(
    font, text="4x4 MIDI Messenger", color=0xFFFFFF)
main_area.anchor_point = (0.5, 0.0)
main_area.anchored_position = (display.width / 2, 0)

# MIDI message number text
msg_area = label.Label(
    font, text="CC Msg: 10", color=0xFFFFFF)
msg_area.anchor_point = (0.0, 0.5)
msg_area.anchored_position = (0, display.height / 2)

# MIDI message value text
val_area = label.Label(
    font, text="CC Val: 50", color=0xFFFFFF)
val_area.anchor_point = (0.0, 1.0)
val_area.anchored_position = (0, display.height)

# MIDI message status text
status_area = label.Label(
    font, text="Sent!", color=0xFFFFFF)
status_area.anchor_point = (1.0, 1.0)
status_area.anchored_position = (display.width, display.height)

splash.append(main_area)
splash.append(msg_area)
splash.append(val_area)
splash.append(status_area)

# MIDI over UART setup for MIDI FeatherWing
uart = busio.UART(board.TX, board.RX, baudrate=31250, timeout=0.001)
midi = adafruit_midi.MIDI(
    midi_in=uart,
    midi_out=uart,
    in_channel=(midi_in_channel - 1),
    out_channel=(midi_out_channel - 1),
    debug=False,
)

# quad rotary encoder setup
ss0 = adafruit_seesaw.seesaw.Seesaw(i2c, 0x49)
ss1 = adafruit_seesaw.seesaw.Seesaw(i2c, 0x4A)
ss2 = adafruit_seesaw.seesaw.Seesaw(i2c, 0x4B)
ss3 = adafruit_seesaw.seesaw.Seesaw(i2c, 0x4C)

# button pins for the encoders
pins = [12, 14, 17, 9]

# interrupts for the button pins. pins are passed as a bitmask
ss0.set_GPIO_interrupts(1 << pins[0] | 1 << pins[1] | 1 << pins[2] | 1 << pins[3], True)
ss1.set_GPIO_interrupts(1 << pins[0] | 1 << pins[1] | 1 << pins[2] | 1 << pins[3], True)
ss2.set_GPIO_interrupts(1 << pins[0] | 1 << pins[1] | 1 << pins[2] | 1 << pins[3], True)
ss3.set_GPIO_interrupts(1 << pins[0] | 1 << pins[1] | 1 << pins[2] | 1 << pins[3], True)

# arrays for the encoders and switches
enc0 = []
enc1 = []
enc2 = []
enc3 = []
sw0 = []
sw1 = []
sw2 = []
sw3 = []

# creating encoders and switches, enabling interrupts for encoders
for i in range(4):
    enc0.append(adafruit_seesaw.rotaryio.IncrementalEncoder(ss0, i))
    enc1.append(adafruit_seesaw.rotaryio.IncrementalEncoder(ss1, i))
    enc2.append(adafruit_seesaw.rotaryio.IncrementalEncoder(ss2, i))
    enc3.append(adafruit_seesaw.rotaryio.IncrementalEncoder(ss3, i))

sw0.append(adafruit_seesaw.digitalio.DigitalIO(ss0, pins[i]))
s0[i].switch_to_input(digitalio.Pull.UP)
sw1.append(adafruit_seesaw.digitalio.DigitalIO(ss1, pins[i]))
s1[i].switch_to_input(digitalio.Pull.UP)
sw2.append(adafruit_seesaw.digitalio.DigitalIO(ss2, pins[i]))
s2[i].switch_to_input(digitalio.Pull.UP)
sw3.append(adafruit_seesaw.digitalio.DigitalIO(ss3, pins[i]))
s3[i].switch_to_input(digitalio.Pull.UP)

ss0.enable_encoder_interrupt(encoder=i)
ss1.enable_encoder_interrupt(encoder=i)
ss2.enable_encoder_interrupt(encoder=i)
ss3.enable_encoder_interrupt(encoder=i)

# neopixels on each PCB
pix0 = adafruit_seesaw.neopixel.NeoPixel(ss0, 18, 4, auto_write = True)
pix0.brightness = 0.5
pix1 = adafruit_seesaw.neopixel.NeoPixel(ss1, 18, 4, auto_write = True)
pix1.brightness = 0.5
pix2 = adafruit_seesaw.neopixel.NeoPixel(ss2, 18, 4, auto_write = True)
pix2.brightness = 0.5
pix3 = adafruit_seesaw.neopixel.NeoPixel(ss3, 18, 4, auto_write = True)
pix3.brightness = 0.5

# onboard Feather neopixel
pix_feather = neopixel.NeoPixel(board.NEOPIXEL, 1, auto_write = True)
pix_feather.brightness = 0.5

# encoder position arrays
last_pos0 = [60, 60, 60, 60]
last_pos1 = [60, 60, 60, 0]
last_pos2 = [0, 0, 0, 120]
last_pos3 = [0, 0, 0, 0]
pos0 = [60, 60, 60, 60]
pos1 = [60, 60, 60, 0]
pos2 = [0, 0, 0, 120]
pos3 = [0, 0, 0, 0]

# color arrays for the neopixels
c0 = [0, 16, 32, 48]
c1 = [64, 80, 96, 112]
c2 = [128, 144, 160, 176]
c3 = [192, 208, 224, 240]

# setting starting colors for neopixels
for r in range(4):
    pix0[r] = colorwheel(c0[r])
    pix1[r] = colorwheel(c1[r])
    pix2[r] = colorwheel(c2[r])
    pix3[r] = colorwheel(c3[r])

# feather neopixel color
_c_feather = 0
pix_feather[0] = colorwheel(_c_feather)

# array of all 16 encoder positions
encoder_positions = [60, 60, 60, 60, 60, 60, 60, 60, 0, 0, 0, 120, 0, 0, 0, 0]

class MIDI_Messages:
    # tracks sending a message and index 0-15
    def __init__(self):
        self.send_msg = False
        self.midi_index = 0

class NeoPixel_Attributes:
    # tracks color, neopixel index and seesaw
    def __init__(self):
        self.color = c0
self.index = 0
self.strip = pix0
self.feather_color = c_feather

async def send_midi(midi_msg):
    # sends MIDI message if send_msg is True/button pressed
    while True:
        if midi_msg.send_msg is True:
            m = midi_msg.midi_index
            main_area.text = f"{cc_values[m]['cc_name']}"
            msg_area.text = f"CC Msg: {cc_values[m]['cc_message']}"
            val_area.text = f"CC Val: {encoder_posititions[m]}"
            midi.send(ControlChange(cc_values[m]['cc_message'], encoder_posititions[m]))
            status_area.text = "Sent!"
            print(f"sending midi: {m}, {encoder_posititions[m]}, {cc_values[m]['cc_message']}"
            time.sleep(1)
            midi_msg.send_msg = False
        else:
            status_area.text = ""
            await asyncio.sleep(0)

async def rainbows(the_color):
    # Updates colors of the neopixels to scroll through rainbow
    while True:
        the_color.feather_color += 8
        the_color.strip[the_color.index] = colorwheel(the_color.color[the_color.index])
        pix_feather[0] = colorwheel(the_color.feather_color)
        await asyncio.sleep(0)

async def monitor_interrupts(pin0, pin1, pin2, pin3, the_color, midi_msg):
    # function to keep encoder value pinned between CC value range
    def normalize(val, min_v, max_v):
        return max(min(max_v, val), min_v)
    # read encoder function
    def read_encoder(enc_group, pos, last_pos, pix, colors, index_diff):
        # check all four encoders if interrupt is detected
        for p in range(4):
            pos[p] = enc_group[p].position
            if pos[p] != last_pos[p]:
                main_index = p + index_diff
                # update CC value
                if pos[p] > last_pos[p]:
                    colors[p] += 8
                    encoder_posititions[main_index] += 1
                else:
                    colors[p] -= 8
                    encoder_posititions[main_index] -= 1
                encoder_posititions[main_index] = normalize(encoder_posititions[main_index],
                                                            cc_values[main_index]['cc_val'][0],
                                                            cc_values[main_index]['cc_val'][1])
            print(main_index, encoder_posititions[main_index],
                  cc_values[main_index]['cc_val'][0],
                  cc_values[main_index]['cc_val'][1])
            colors[p] = (colors[p] + 256) % 256
            # wrap around to 0-256
            print(main_index, encoder_posititions[main_index])
            main_area.text = f"{cc_values[main_index]['cc_name']}"
            msg_area.text = f"CC Msg: {cc_values[main_index]['cc_message']}"
            val_area.text = f"CC Val: {encoder_posititions[main_index]}"
            last_pos[p] = pos[p]
        # update NeoPixel colors
        the_color.color = colors
        the_color.index = p
        the_color.strip = pix

    # function to read button press
    def press_switches(sw, index):
        ...
if not sw[index].value:
    # signals that a MIDI message should be sent
    midi_msg.send_msg = True
    midi_msg.midi_index = index
    print(f"button {index} pressed")
# interrupt pins are passed as a keypad
    with keypad.Keys(
        (pin0, pin1, pin2, pin3,), value_when_pressed=False, pull=True
    ) as keys:
        while True:
            key_event = keys.events.get()
            if key_event and key_event.pressed:
                key_number = key_event.key_number
                # seesaw 0
                if key_number == 0:
                    read_encoder(enc0, pos0, last_pos0, pix0, c0, 0)
                    press_switches(sw0, 0)
                    press_switches(sw0, 1)
                    press_switches(sw0, 2)
                    press_switches(sw0, 3)
                # seesaw 1
                elif key_number == 1:
                    read_encoder(enc1, pos1, last_pos1, pix1, c1, 4)
                    press_switches(sw1, 0)
                    press_switches(sw1, 1)
                    press_switches(sw1, 2)
                    press_switches(sw1, 3)
                    # update index to 4-7
                    midi_msg.midi_index = midi_msg.midi_index + 4
                # seesaw 2
                elif key_number == 2:
                    read_encoder(enc2, pos2, last_pos2, pix2, c2, 8)
                    press_switches(sw2, 0)
                    press_switches(sw2, 1)
                    press_switches(sw2, 2)
                    press_switches(sw2, 3)
                    # update index 8-11
                    midi_msg.midi_index = midi_msg.midi_index + 8
                # seesaw 3
                else:
                    read_encoder(enc3, pos3, last_pos3, pix3, c3, 12)
                    press_switches(sw3, 0)
                    press_switches(sw3, 1)
                    press_switches(sw3, 2)
                    press_switches(sw3, 3)
                    # update index 12-15
                    midi_msg.midi_index = midi_msg.midi_index + 12
                # clear interrupt flag to reset interrupt pin
                ss0.get_GPIO_interrupt_flag()
                ss1.get_GPIO_interrupt_flag()
                ss2.get_GPIO_interrupt_flag()
                ss3.get_GPIO_interrupt_flag()
                await asyncio.sleep(0)

async def main():
    the_color = NeoPixel_Attributes()
    midi_msg = MIDI_Messages()
    # interrupt listener task
    interrupt_task = asyncio.create_task(monitor_interrupts(board.D5, board.D6,
                                                            board.D9,
                                                            board.D10, the_color,
                                                            midi_msg))
    # neopixel task
    pixels_task = asyncio.create_task(rainbows(the_color))
    # midi task
    midi_task = asyncio.create_task(send_midi(midi_msg))

    await asyncio.gather(interrupt_task, pixels_task, midi_task)
asyncio.run(main())

Upload the Code and Libraries to the Feather RP2040

After downloading the Project Bundle, plug your Feather RP2040 into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called CIRCUITPY. Unzip the folder and copy the following items to the Feather RP2040's CIRCUITPY drive:

- lib folder
- code.py
- OCRA_small.pcf

Your Feather RP2040 CIRCUITPY drive should look like this after copying the lib folder, the font file and the code.py file:

---

How the CircuitPython Code Works

This code uses `asyncio.run(main())`, which lets you queue up tasks that take turns running. Combined with using the interrupt pins on the rotary encoder breakouts, this makes reading 16 rotary encoders and button presses over I2C much faster than if you were to try and do it with a traditional single `while True:` loop.

The code begins with a dictionary that contains the MIDI CC information assigned to each rotary encoder. You can edit this for whatever device you are controlling.
l has the minimum and maximum CC values, **cc_message** has the MIDI CC message number and **cc_name** has the name of each command. The **cc_name** is displayed on the OLED.

```python
# MIDI CC messages, values and names assigned to each encoder
cc_values = [
    {'cc_val': (0, 127), 'cc_message': (14), 'cc_name': "Volume"},
    {'cc_val': (0, 127), 'cc_message': (15), 'cc_name': "Repeats"},
    {'cc_val': (0, 127), 'cc_message': (16), 'cc_name': "Size"},
    {'cc_val': (0, 127), 'cc_message': (17), 'cc_name': "Mod"},
    {'cc_val': (0, 127), 'cc_message': (18), 'cc_name': "Spread"},
    {'cc_val': (0, 127), 'cc_message': (19), 'cc_name': "Scan"},
    {'cc_val': (0, 127), 'cc_message': (20), 'cc_name': "Ramp"},
    {'cc_val': (1, 3), 'cc_message': (21), 'cc_name': "Mod Number"},
    {'cc_val': (1, 3), 'cc_message': (22), 'cc_name': "Mod Bank"},
    {'cc_val': (1, 3), 'cc_message': (23), 'cc_name': "Mode"},
    {'cc_val': (0, 1), 'cc_message': (102), 'cc_name': "Bypass/Engage"},
    {'cc_val': (60, 200), 'cc_message': (93), 'cc_name': "Tap Tempo"},
    {'cc_val': (0, 1), 'cc_message': (24), 'cc_name': "Loop (R Hold)"},
    {'cc_val': (0, 1), 'cc_message': (25), 'cc_name': "Scan (L Hold)"},
    {'cc_val': (0, 127), 'cc_message': (26), 'cc_name': "Clear (Both Hold)"},
    {'cc_val': (0, 1), 'cc_message': (51), 'cc_name': "MIDI Clock Ignore"}
]
```

**Display**

Next is the display setup for the OLED. There are four text elements. The **main_area** will show the name of the CC message that you have selected. The **msg_area** shows the MIDI CC message number and the **val_area** shows the CC value that is currently assigned to the rotary encoder. **status_area** will show "**Sent!**" when you send the MIDI CC message.

```python
# STEMMA OLED setup
display_bus = displayio.I2CDisplay(i2c, device_address=0x3D, reset=oled_reset)
display = adafruit_displayio_ssd1306.SSD1306(display_bus, width=128, height=64)

splash = displayio.Group()
display.root_group = splash
font = bitmap_font.load_font('/OCRA_small.pcf')
# main label/MIDI message name text; centered
main_area = label.Label(  
    font, text="4x4 MIDI Messenger", color=0xFFFFFF)
main_area.anchor_point = (0.5, 0.0)
main_area.anchored_position = (display.width / 2, 0)
# MIDI message number text
msg_area = label.Label(  
    font, text="CC Msg: 10", color=0xFFFFFF)
msg_area.anchor_point = (0.0, 0.5)
msg_area.anchored_position = (0, display.height / 2)
# MIDI message value text
val_area = label.Label(  
    font, text="CC Val: 50", color=0xFFFFFF)
val_area.anchor_point = (0.0, 1.0)
val_area.anchored_position = (0, display.height)
# MIDI message status text
status_area = label.Label(  
    font, text="Sent!", color=0xFFFFFF)
status_area.anchor_point = (1.0, 1.0)
status_area.anchored_position = (display.width, display.height)
```
The MIDI FeatherWing uses MIDI over UART to send and receive MIDI messages.

```python
import busio
import adafruit_midi

# MIDI over UART setup for MIDI FeatherWing
uart = busio.UART(board.TX, board.RX, baudrate=31250, timeout=0.001)
midi_in_channel = 1
midi_out_channel = 1
midi = adafruit_midi.MIDI(
    midi_in=uart,
    midi_out=uart,
    in_channel=(midi_in_channel - 1),
    out_channel=(midi_out_channel - 1),
    debug=False,
)
```

seesaw Setup

Four seesaw objects are created for each of the quad rotary encoder breakouts. Each of the breakouts use the same pins for the switches on the rotary encoders (12, 14, 17 and 9).

```python
# quad rotary encoder setup
ss0 = adafruit_seesaw.seesaw.Seesaw(i2c, 0x49)
ss1 = adafruit_seesaw.seesaw.Seesaw(i2c, 0x4A)
ss2 = adafruit_seesaw.seesaw.Seesaw(i2c, 0x4B)
ss3 = adafruit_seesaw.seesaw.Seesaw(i2c, 0x4C)
# button pins for the encoders
pins = [12, 14, 17, 9]
```

Interrupts

Interrupts are setup for the switch pins by passing the pins as a bitmask. As a result, whenever you engage one of the switches, you'll see the INT LED turn on on the breakout.

```python
# interrupts for the button pins. pins are passed as a bitmask
ss0.set_GPIO_interrupts(1 &lt;&lt; pins[0] | 1 &lt;&lt; pins[1] | 1 &lt;&lt; pins[2] | 1 &lt;&lt; pins[3], True)
ss1.set_GPIO_interrupts(1 &lt;&lt; pins[0] | 1 &lt;&lt; pins[1] | 1 &lt;&lt; pins[2] | 1 &lt;&lt; pins[3], True)
ss2.set_GPIO_interrupts(1 &lt;&lt; pins[0] | 1 &lt;&lt; pins[1] | 1 &lt;&lt; pins[2] | 1 &lt;&lt; pins[3], True)
ss3.set_GPIO_interrupts(1 &lt;&lt; pins[0] | 1 &lt;&lt; pins[1] | 1 &lt;&lt; pins[2] | 1 &lt;&lt; pins[3], True)
```
Encoders, Switches and More Interrupts

A `for` statement instantiates four encoders and four switches on each of the breakouts. Interrupts are also enabled on each of the encoders.

```python
# arrays for the encoders and switches
enc0 = []
enc1 = []
enc2 = []
enc3 = []
sw0 = []
sw1 = []
sw2 = []
sw3 = []
# creating encoders and switches, enabling interrupts for encoders
for i in range(4):
    enc0.append(adafruit_seesaw.rotaryio.IncrementalEncoder(ss0, i))
    enc1.append(adafruit_seesaw.rotaryio.IncrementalEncoder(ss1, i))
    enc2.append(adafruit_seesaw.rotaryio.IncrementalEncoder(ss2, i))
    enc3.append(adafruit_seesaw.rotaryio.IncrementalEncoder(ss3, i))
    sw0.append(adafruit_seesaw.digitalio.DigitalIO(ss0, pins[i]))
    sw0[i].switch_to_input(digitalio.Pull.UP)
    sw1.append(adafruit_seesaw.digitalio.DigitalIO(ss1, pins[i]))
    sw1[i].switch_to_input(digitalio.Pull.UP)
    sw2.append(adafruit_seesaw.digitalio.DigitalIO(ss2, pins[i]))
    sw2[i].switch_to_input(digitalio.Pull.UP)
    sw3.append(adafruit_seesaw.digitalio.DigitalIO(ss3, pins[i]))
    sw3[i].switch_to_input(digitalio.Pull.UP)
    ss0.enable_encoder_interrupt(encoder=i)
    ss1.enable_encoder_interrupt(encoder=i)
    ss2.enable_encoder_interrupt(encoder=i)
    ss3.enable_encoder_interrupt(encoder=i)
```

NeoPixels

The four NeoPixels on each breakout are instantiated. Then, each NeoPixel is assigned a color value from its corresponding color array.

```python
# neopixels on each PCB
pix0 = adafruit_seesaw.neopixel.NeoPixel(ss0, 18, 4, auto_write = True)
pix0.brightness = 0.5
pix1 = adafruit_seesaw.neopixel.NeoPixel(ss1, 18, 4, auto_write = True)
pix1.brightness = 0.5
pix2 = adafruit_seesaw.neopixel.NeoPixel(ss2, 18, 4, auto_write = True)
pix2.brightness = 0.5
pix3 = adafruit_seesaw.neopixel.NeoPixel(ss3, 18, 4, auto_write = True)
pix3.brightness = 0.5

# color arrays for the neopixels
c0 = [0, 16, 32, 48]
c1 = [64, 80, 96, 112]
c2 = [128, 144, 160, 176]
c3 = [192, 208, 224, 240]
# setting starting colors for neopixels
for r in range(4):
    pix0[r] = colorwheel(c0[r])
    pix1[r] = colorwheel(c1[r])
```
Tracking Encoder Positions

Each encoder position and last position are tracked in arrays. There is also one large array that tracks all sixteen encoders.

```python
# encoder position arrays
last_pos0 = [60, 60, 60, 60]
last_pos1 = [60, 60, 60, 0]
last_pos2 = [0, 0, 0, 120]
last_pos3 = [0, 0, 0, 0]
pos0 = [60, 60, 60, 60]
pos1 = [60, 60, 60, 0]
pos2 = [0, 0, 0, 120]
pos3 = [0, 0, 0, 0]
# array of all 16 encoder positions
encoder_positions = [60, 60, 60, 60, 60, 60, 60, 60, 0, 0, 0, 120, 0, 0, 0, 0]
```

Classes

There are two classes that are used for the asyncio portion of the code. These classes hold values that are passed between the async tasks. The `MIDI_Messages` class tracks the selected index and whether or not a message should be sent. The `NeoPixel_Attributes` class tracks the current color, index and NeoPixel object.

```python
class MIDI_Messages:
    # tracks sending a message and index 0-15
    def __init__(self):
        self.send_msg = False
        self.midi_index = 0

class NeoPixel_Attributes:
    # tracks color, neopixel index and seesaw
    def __init__(self):
        self.color = c0
        self.index = 0
        self.strip = pix0
```

Tasks

There are three async tasks. `send_midi` sends a MIDI message if the `send_msg` attribute is `True`.

```python
async def send_midi(midi_msg):
    # sends send_msg if send_msg is True/button pressed
    while True:
        if midi_msg.send_msg is True:
            m = midi_msg.midi_index
            main_area.text = f"{cc_values[m]['cc_name']}"
```
msg_area.text = f"CC Msg: {cc_values[m]['cc_message']}"
val_area.text = f"CC Val: {encoder_posititions[m]}"
midi.send(ControlChange(cc_values[m]['cc_message'],
encoder_posititions[m]))
status_area.text = "Sent!"
print(f"sending midi: {m}, {encoder_posititions[m]}, {cc_values[m] ['cc_message']}")
time.sleep(1)
midi_msg.send_msg = False
else:
    status_area.text = ""
    await asyncio.sleep(0)

async def rainbows(the_color):
    # Updates colors of the neopixels to scroll through rainbow
    while True:
        the_color.strip[the_color.index] =
        colorwheel(the_color.color[the_color.index])
        await asyncio.sleep(0)

Rainbows changes the color of the NeoPixels.

The main task is monitor_interrupts(). The pins that the interrupt pins are
attached to are passed as Keypad keys. If an interrupt is triggered, then I2C is
scanned on the corresponding seesaw board for changes in the rotary encoder
position or the switch status.

The rotary encoders change the MIDI CC value and engaging a switch triggers
sending a MIDI message with send_midi(). These values are displayed on the
OLED. The selected encoder or switch corresponds with the index tracked by the MIDI
I_Messages class.

Before the task ends, the interrupt flag is reset with get_GPIO_interrupt_flag() for each seesaw.

async def monitor_interrupts(pin0, pin1, pin2, pin3, the_color, midi_msg):
    # function to keep encoder value pinned between CC value range
    def normalize(val, min_v, max_v):
        return max(min(max_v, val), min_v)
    # read encoder function
    def read_encoder(enc_group, pos, last_pos, pix, colors, index_diff):
        # check all four encoders if interrupt is detected
        for p in range(4):
            pos[p] = enc_group[p].position
            if pos[p] != last_pos[p]:
                main_index = p + index_diff
                # update CC value
                if pos[p] > last_pos[p]:
                    colors[p] += 8
                    encoder_posititions[main_index] =
                    encoder_posititions[main_index] + 1
                else:
                    colors[p] -= 8
                    encoder_posititions[main_index] =
                    encoder_posititions[main_index] - 1
                encoder_posititions[main_index] =
normalize(encoder_posititions[main_index],
 cc_values[main_index]['cc_val'][0],
 cc_values[main_index]['cc_val'][1])

colors[p] = (colors[p] + 256) % 256  # wrap around to 0-256
print(main_index, encoder_posititions[main_index])
main_area.text = f"{cc_values[main_index]['cc_name']}"
msg_area.text = f"CC Msg: {cc_values[main_index]['cc_message']}"
val_area.text = f"CC Val: {encoder_posititions[main_index]}"
last_pos[p] = pos[p]
# update NeoPixel colors
the_color.color = colors
the_color.index = p
the_color.strip = pix

# function to read button press
def press_switches(sw, index):
    if not sw[index].value:
        # signals that a MIDI message should be sent
        midi_msg.send_msg = True
        midi_msg.midi_index = index
        print(f"button {index} pressed")

    # interrupt pins are passed as a keypad
    with keypad.Keys((pin0, pin1, pin2, pin3,), value_when_pressed=False, pull=True
                     ) as keys:
        while True:
            key_event = keys.events.get()
            if key_event and key_event.pressed:
                key_number = key_event.key_number
                # seesaw 0
                if key_number == 0:
                    read_encoder(enc0, pos0, last_pos0, pix0, c0, 0)
                    press_switches(sw0, 0)
                    press_switches(sw0, 1)
                    press_switches(sw0, 2)
                    press_switches(sw0, 3)
                # seesaw 1
                elif key_number == 1:
                    read_encoder(enc1, pos1, last_pos1, pix1, c1, 4)
                    press_switches(sw1, 0)
                    press_switches(sw1, 1)
                    press_switches(sw1, 2)
                    press_switches(sw1, 3)
                    # update index to 4-7
                    midi_msg.midi_index = midi_msg.midi_index + 4
                # seesaw 2
                elif key_number == 2:
                    read_encoder(enc2, pos2, last_pos2, pix2, c2, 8)
                    press_switches(sw2, 0)
                    press_switches(sw2, 1)
                    press_switches(sw2, 2)
                    press_switches(sw2, 3)
                    # update index 8-11
                    midi_msg.midi_index = midi_msg.midi_index + 8
                # seesaw 3
                else:
                    read_encoder(enc3, pos3, last_pos3, pix3, c3, 12)
                    press_switches(sw3, 0)
                    press_switches(sw3, 1)
                    press_switches(sw3, 2)
                    press_switches(sw3, 3)
                    # update index 12-15
                    midi_msg.midi_index = midi_msg.midi_index + 12
    # clear interrupt flag to reset interrupt pin
    ss0.get_GPIO_interrupt_flag()
    ss1.get_GPIO_interrupt_flag()
    ss2.get_GPIO_interrupt_flag()
    ss3.get_GPIO_interrupt_flag()
    await asyncio.sleep(0)
Run!

In the `main()` function, the tasks are created and gathered. After that, they are run in a loop with `run()`.

```python
async def main():
    the_color = NeoPixel_Attributes()
    midi_msg = MIDI_Messages()
    # interrupt listener task
    interrupt_task = asyncio.create_task(monitor_interrupts(board.D5, board.D6, board.D9, board.D10, the_color, midi_msg))
    # neopixel task
    pixels_task = asyncio.create_task(rainbows(the_color))
    # midi task
    midi_task = asyncio.create_task(send_midi(midi_msg))

    await asyncio.gather(interrupt_task, pixels_task, midi_task)

asyncio.run(main())
```

Soldering

Headers

Solder pin headers to the Feather RP2040 and MIDI FeatherWing. Solder socket headers to the FeatherWing Doubler.
Encoders

Solder rotary encoders to the four quad rotary encoder breakouts for a total of 16 rotary encoders. You can use the optional 3D printed NeoPixel riser described on the 3D Printing page to diffuse the NeoPixel underneath the rotary encoder.

I2C Address Jumpers

The quad rotary encoder breakouts need to have their address jumpers cut to change the I2C address of each board:

- Breakout 0: default address 0x49 (no jumpers cut)
- Breakout 1: address 0x4A (jumper A0 cut)
- Breakout 2: address 0x4B (jumper A1 cut)
- Breakout 3: address 0x4C (jumpers A0 and A1 cut)

Interrupt Pins

Cut, splice and tin four pieces of wire. These wires will be used to connect the interrupt pins on the quad rotary encoder breakouts to the Feather RP2040 on the FeatherWing Doubler. Solder the wires to the following pins:

- D5 (blue wire)
- D6 (green wire)
- D9 (white wire)
- D10 (yellow wire)
Solder the other ends of the wires to the INT pins on the quad rotary encoder breakouts.

Pin D5 to breakout 0 (0x49) INT (blue wire)
Pin D6 to breakout 1 (0x4A) INT (green wire)
Pin D9 to breakout 2 (0x4B) INT (white wire)
Pin D10 to breakout 3 (0x4C) INT (yellow wire)

STEMMA QT Cables

The rest of the wiring is done using STEMMA QT cables. The quad rotary encoders and OLED display are daisy chained to the Feather RP2040.

Feather STEMMA port to breakout 0 (0x49)
Breakout 0 (0x49) to breakout 1 (0x4A)
Breakout 1 (0x4A) to breakout 2 (0x4B)
Breakout 2 (0x4B) to breakout 3 (0x4C)
Breakout 3 (0x4C) to OLED display

Assembly

Mount the FeatherWing Doubler

Insert 8 M2.5 screws into the mounting holes at the bottom of the 3D printed case.
Insert the FeatherWing Doubler over the 8 screws and secure it in place with 8 M2.5 nuts.

Plug in the Feather RP2040 and MIDI FeatherWing into the FeatherWing Doubler.
Mount the OLED

Attach the OLED display to the cutout on the 3D printed lid. Secure the display with M2.5 screws and nuts in the mounting holes.

Mount the Rotary Encoders

Place the washers on each of the rotary encoders to prep them for mounting.
Insert the rotary encoders into the mounting holes on the 3D printed lid. Breakout 0 (0x49) should be at the top and breakout 3 (0x4C) should be at the bottom.

Secure the rotary encoders with the included nuts. Close up the case by attaching the lid with M3 screws into the mounting holes in the corners.

Knobs

Finish the assembly by attaching knobs to the rotary encoders.
Use

Plug a 5-DIN MIDI cable into the out port on the MIDI FeatherWing. Then, plug in a USB-C cable to the Feather RP2040 to power up the project. After connecting the MIDI messenger to an external piece of MIDI gear, such as a guitar pedal, you'll be able to send MIDI CC messages to the gear to control it.

At the top of the code, you can customize the MIDI output channel and the MIDI CC value range, message number and message name.
You can turn the rotary encoders to set different CC values. When you're ready to send the CC message, you can press the switch on the rotary encoder (push down for a click to engage the switch, release to reset).