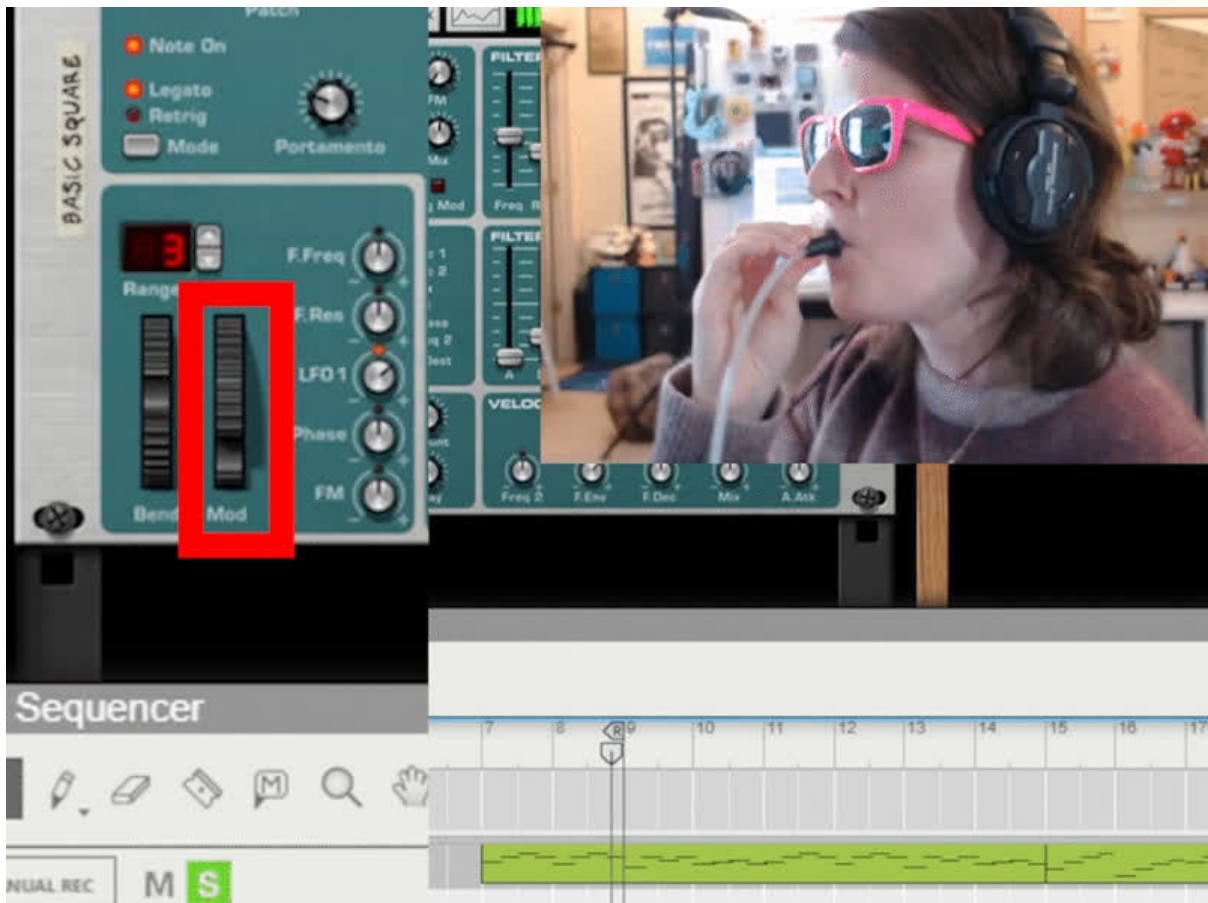




MIDI Breath Controller

Created by Liz Clark



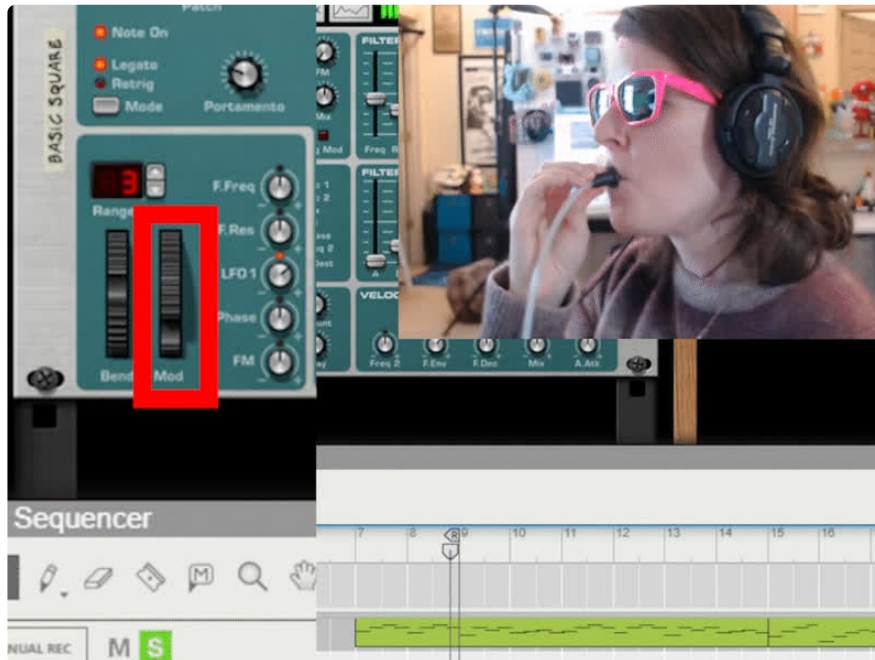
<https://learn.adafruit.com/midi-breath-controller>

Last updated on 2026-03-04 11:07:36 PM UTC

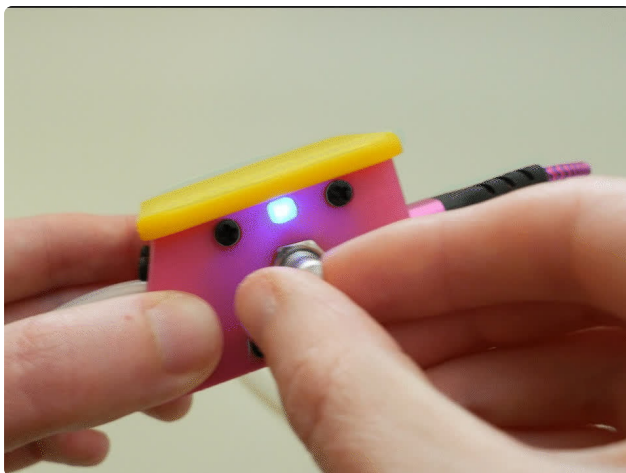
Table of Contents

Overview	3
<ul style="list-style-type: none">• Parts	
Circuit Diagram	6
3D Printing	6
Install CircuitPython	8
<ul style="list-style-type: none">• CircuitPython Quickstart• Safe Mode• Flash Resetting UF2	
Code the Controller	12
<ul style="list-style-type: none">• Upload the Code and Libraries to the QT Py RP2040• How the CircuitPython Code Works	
Assembly	18
Use	21
<ul style="list-style-type: none">• Customize the Code	

Overview

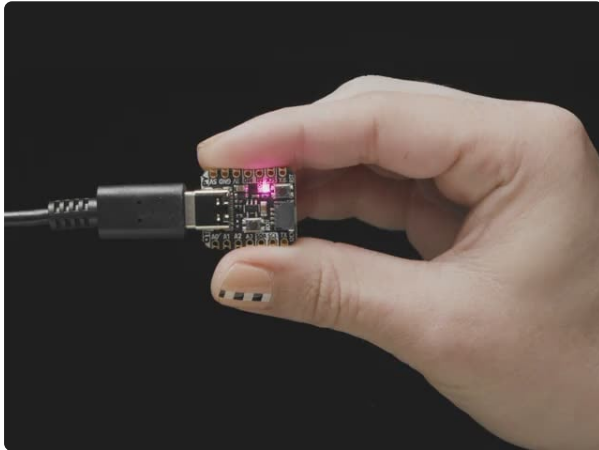


Deep breathing isn't only for mindfulness, it's for sending MIDI CC messages too. You can build a MIDI breath controller with a BMP585 air pressure sensor. This particular sensor has a port that you can attach a tube to for targeted pressure readings. In this scenario, the sensor is attached to a QT Py RP2040 running CircuitPython code that maps a pressure reading range to MIDI CC message values.



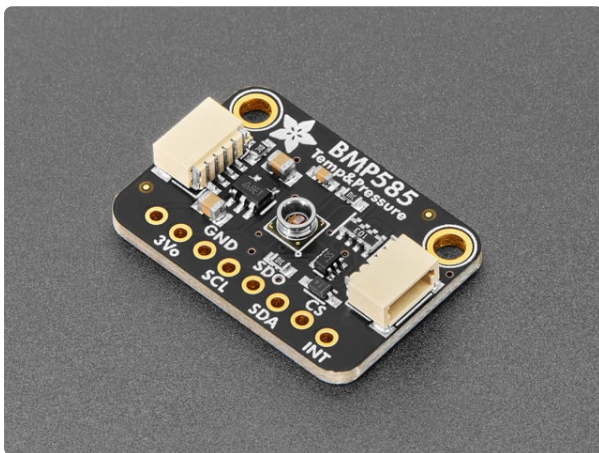
Use the rotary encoder to switch which MIDI CC message is sent from the controller. The NeoPixel LED light acts as an indicator.

Parts



[Adafruit QT Py RP2040](https://www.adafruit.com/product/4900)

What a cutie pie! Or is it... a QT Py? This diminutive dev board comes with one of our new favorite chip, the RP2040. It's been made famous in the new <https://www.adafruit.com/product/4900>



[Adafruit BMP585 Ported I2C / SPI Temperature and Pressure Sensor](https://www.adafruit.com/product/6413)

Bosch has been a leader in barometric pressure sensors, from the BMP085. <https://www.adafruit.com/product/6413>



[Adafruit I2C Stemma QT Rotary Encoder Breakout with Encoder](https://www.adafruit.com/product/5880)

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/5880>



Potentiometer Knob - Soft Touch T18 - White

Oh say can you see
By the knob's early light...
Sorry - we thought that...

<https://www.adafruit.com/product/2047>

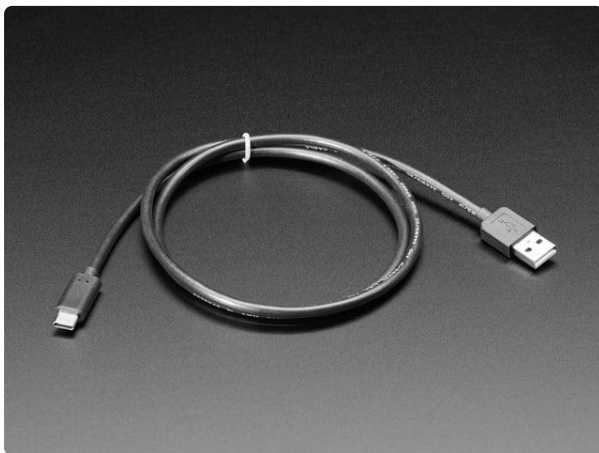
Two Stemma QT Cables



STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long

This 4-wire cable is a little over 100mm / 4" long and fitted with JST-SH female 4-pin connectors on both ends. Compared with the chunkier JST-PH these are 1mm pitch instead of...

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



USB Type A to Type C Cable - approx 1 meter / 3 ft long

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

<https://www.adafruit.com/product/4474>

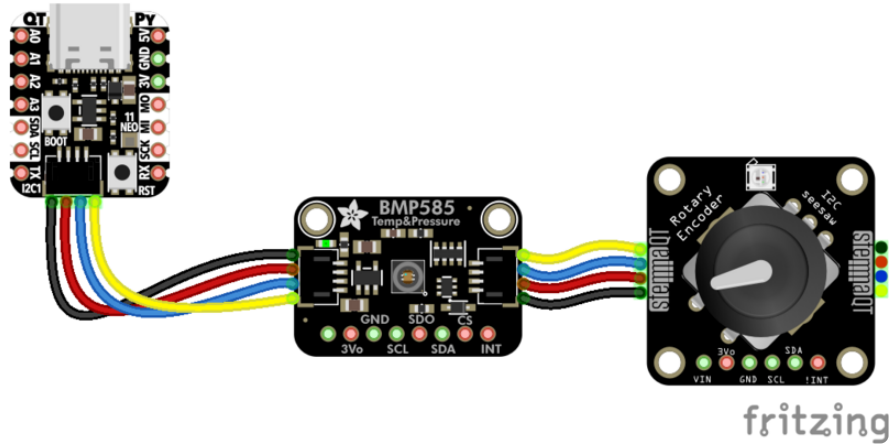
McMaster-Carr sells food grade tubing that works really well for this project:

1 x Odor-Resistant Soft Rubber Tubing for Food and Beverage

1/8" inner diameter, 1/4" outer diameter

<https://www.mcmaster.com/catalog/132/159/9372N7>

Circuit Diagram



All of the electronics connect together with STEMMA QT cables.

Plug the **BMP585** sensor into the STEMMA QT port on the **QT Py RP2040**. Then, plug the **seesaw rotary encoder** into the other side of the **BMP585** sensor.

3D Printing



You can 3D print all of the case parts for this project. The files can be downloaded directly below:

<https://adafru.it/1aAy>



The mouthpiece fits snugly onto the end of the tube.



The QT Py tray is secured inside of the enclosure with screws. There is a cutout in the enclosure for the USB C port on the QT Py.



The small peg is inserted into the side of the case to diffuse the NeoPixel on the rotary encoder. For maximum effect, print it in clear filament.

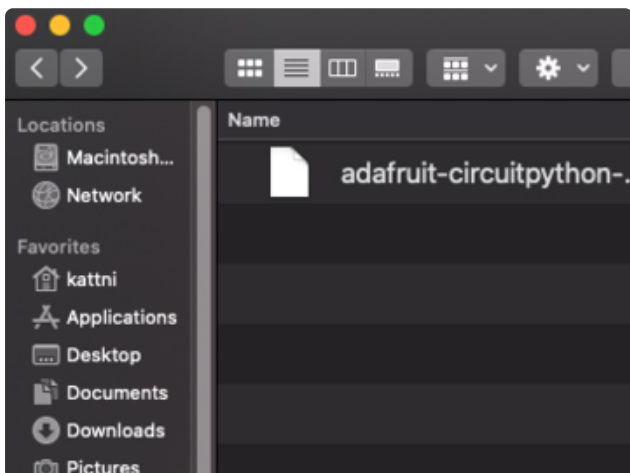
Install CircuitPython

[CircuitPython \(https://adafruit.it/tB7\)](https://adafruit.it/tB7) is a derivative of [MicroPython \(https://adafruit.it/BeZ\)](https://adafruit.it/BeZ) 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

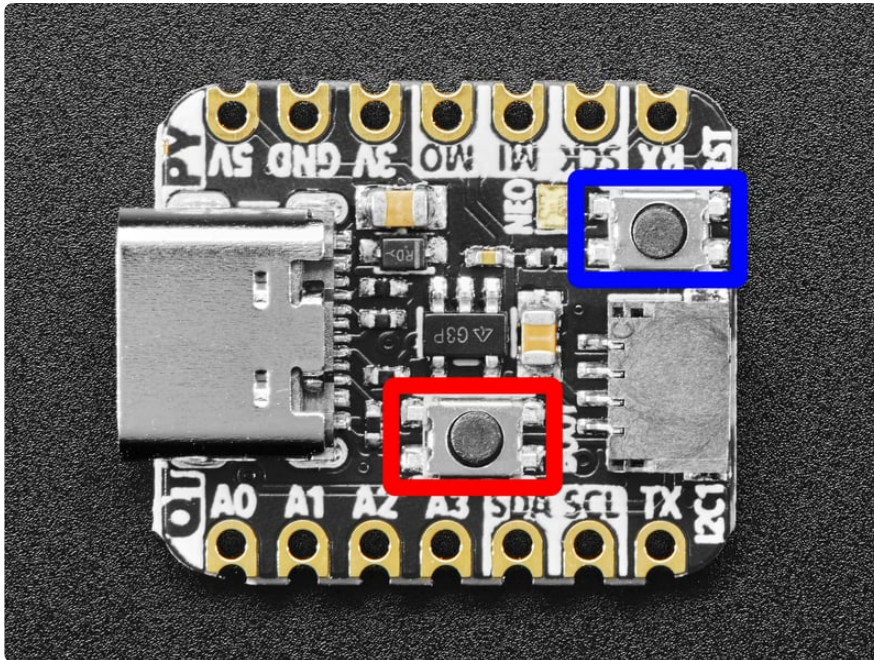
Follow this step-by-step to quickly get CircuitPython running on your board.

<https://adafruit.it/RLD>



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.

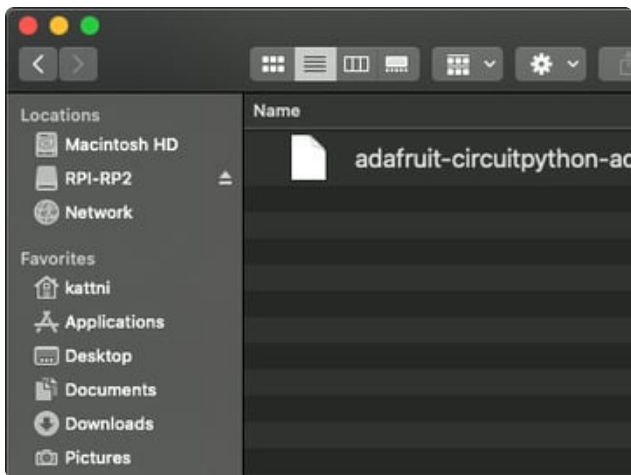


To enter the bootloader, hold down the **BOOT/BOOTSEL** button (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset** button (highlighted in red or blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

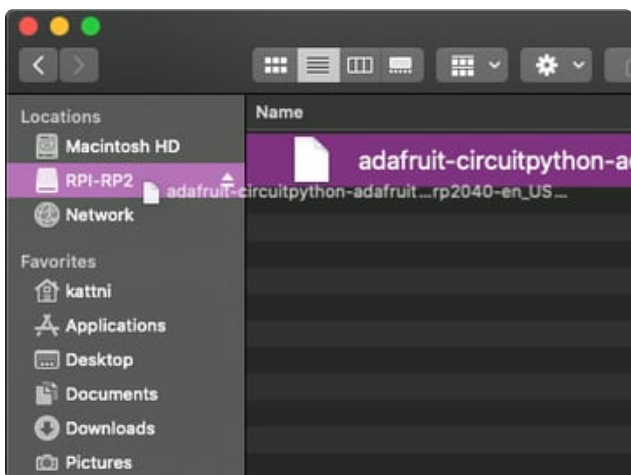
If the drive does not appear, release all the buttons, and then repeat the process above.

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.

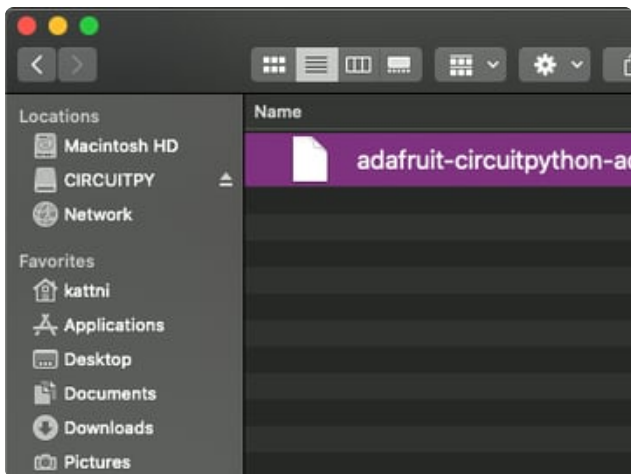
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



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 CIRCUITPY doesn't show up as a disk drive after installing CircuitPython, try loading this 'nuke' UF2 to RPI-RP2. 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.

<https://adafru.it/RLE>

Code the Controller

Once you've finished setting up your QT Py 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 to your computer as a zipped folder.

```
# SPDX-FileCopyrightText: Copyright (c) 2026 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT
'''QT Py RP2040 MIDI Breath Controller with BMP585'''
import time
import board
import simpleio
from adafruit_bmp5xx import BMP5XX
from adafruit_seesaw import digitalio, neopixel, rotaryio, seesaw
import usb_midi
import adafruit_midi
from adafruit_midi.control_change import ControlChange
from adafruit_midi.channel_pressure import ChannelPressure

SEALEVELPRESSURE_HPA = 1013.25
change_sense = 0.1 # change sensitivity
low_press = 995 # lowest pressure reading range
high_press = 1032 # highest pressure reading range

i2c = board.STEMMA_I2C()
bmp = BMP5XX.over_i2c(i2c)
bmp.sea_level_pressure = SEALEVELPRESSURE_HPA

seesaw = seesaw.Seesaw(i2c, addr=0x36)
seesaw.pin_mode(24, seesaw.INPUT_PULLUP)
button = digitalio.DigitalIO(seesaw, 24)
encoder = rotaryio.IncrementalEncoder(seesaw)
pixel = neopixel.NeoPixel(seesaw, 6, 1)
pixel.brightness = 0.2

midi = adafruit_midi.MIDI(midi_out=usb_midi.ports[1], out_channel=0)
# CC message channels
# last index is place holder for Channel Pressure message type
messages = [1, 2, 7, 64, 0]
# neopixel colors to associate with CC messages
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255),
```

```

        (255, 255, 0), (255, 0, 255)]

pressure = 0
mod_val1 = 0
mod_val2 = 0
effect_index = 0
last_position = 0
button_state = False
pixel.fill(colors[effect_index])

while True:
    position = -encoder.position
    if position != last_position:
        # encoder changes midi CC message type
        if position > last_position:
            effect_index = (effect_index + 1) % len(messages)
        else:
            effect_index = (effect_index - 1) % len(messages)
        pixel.fill(colors[effect_index])
        last_position = position
    if not button.value and not button_state:
        pixel.fill((255, 255, 255))
        # button press sends MIDI panic
        panic = ControlChange(123, 0)
        midi.send(panic)
        # and turns sustain off
        sus_off = ControlChange(64, 0)
        midi.send(sus_off)
        button_state = True
    if button.value and button_state:
        # reset neopixel to CC msg color
        pixel.fill(colors[effect_index])
        button_state = False
    if bmp.data_ready:
        # get pressure reading
        pressure = bmp.pressure
        # if the pressure has changed enough
        # (adjust change_sense value at top to inc or dec)
        if abs(pressure - mod_val2) > change_sense:
            # map pressure reading to CC range
            mod_val1 = round(simpleio.map_range(pressure, low_press, high_press, 0,
127))

            # updates previous value to hold current value
            mod_val2 = pressure
            # MIDI data has to be sent as an integer
            modulation = int(mod_val1)
            # possible midi messages determined by effect_index value:
            # 1: modulation
            # 2: breath controller
            # 7: volume
            # 64: sustain
            # ChannelPressure(modulation)
            if effect_index < 4:
                # prep CC message with CC number and value as mapped pressure
reading
                modWheel = ControlChange(messages[effect_index], modulation)
            else:
                # prep Channel Pressure message with value as mapped pressure
reading
                modWheel = ChannelPressure(modulation)
            # CC message is sent
            midi.send(modWheel)
            # print(modWheel)
            # delay to settle MIDI data
            time.sleep(0.001)

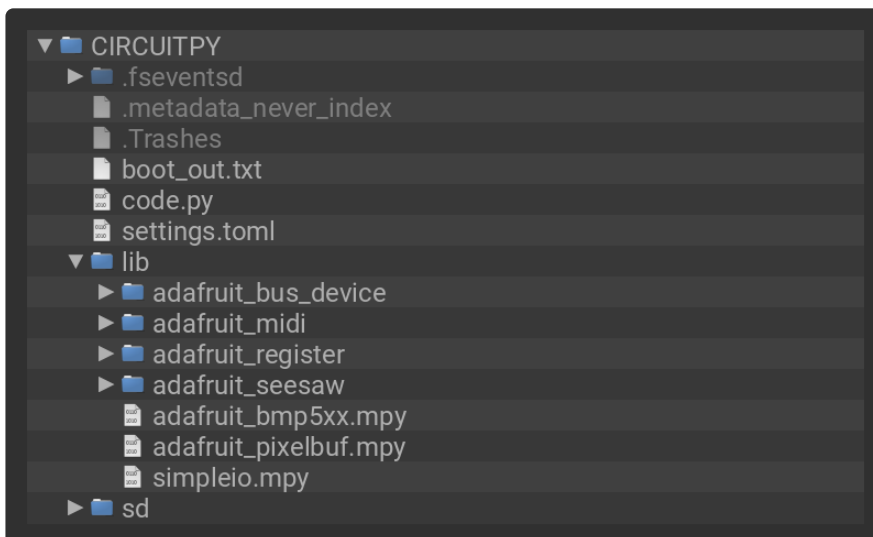
```

Upload the Code and Libraries to the QT Py RP2040

After downloading the Project Bundle, plug your QT Py 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 QT Py RP2040's **CIRCUITPY** drive.

- **lib** folder
- **code.py**

Your QT Py RP2040 **CIRCUITPY** drive should look like this after copying the **lib** folder and **code.py** file:



How the CircuitPython Code Works

At the top of the code are some user parameters that you can edit:

- **SEALEVELPRESSURE_HPA** - the default sea level pressure
- **change_sense** - the difference between pressure readings that needs to be detected before it sends a new MIDI message
- **low_press** - the low range of the pressure readings
- **high_press** - the high range of the pressure readings

You can update the **low_press** and **high_press** values depending on the pressure readings that you get from your use of the pressure sensor.

```
SEALEVELPRESSURE_HPA = 1013.25
change_sense = 0.1 # change sensitivity
low_press = 995 # lowest pressure reading range
high_press = 1032 # highest pressure reading range
```

I2C Devices

Next, the I2C devices are instantiated. First, the BMP585 sensor and then the seesaw rotary encoder, switch and onboard NeoPixel.

```
i2c = board.STEMMA_I2C()
bmp = BMP5XX.over_i2c(i2c)
bmp.sea_level_pressure = SEALEVELPRESSURE_HPA

seesaw = seesaw.Seesaw(i2c, addr=0x36)
seesaw.pin_mode(24, seesaw.INPUT_PULLUP)
button = digitalio.DigitalIO(seesaw, 24)
encoder = rotaryio.IncrementalEncoder(seesaw)
pixel = neopixel.NeoPixel(seesaw, 6, 1)
pixel.brightness = 0.2
```

MIDI and Lists

A MIDI object is created to use USB MIDI out. Two lists are used:

- `messages` - this list has the ControlChange (CC) message numbers
- `colors` - this list has the associated colors for the NeoPixel on the rotary encoder. These colors change along with the CC message.

```
midi = adafruit_midi.MIDI(midi_out=usb_midi.ports[1], out_channel=0)
# CC message channels
# last index is place holder for Channel Pressure message type
messages = [1, 2, 7, 64, 0]
# neopixel colors to associate with CC messages
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255),
          (255, 255, 0), (255, 0, 255)]
```

States and Variables

A few states and variables are used throughout the loop. `pressure` holds the pressure reading from the BMP585. `mod_val1` and `mod_val2` hold the modulation values that are sent as a part of the MIDI CC message. `effect_index` tracks the MIDI CC message from the messages list. `last_position` tracks the previous position for the rotary encoder. `button_state` is the state of the button for debouncing.

Finally, right before the loop, the NeoPixel on the rotary encoder is lit up with the color that corresponds with the default MIDI CC message.

```
pressure = 0
mod_val1 = 0
mod_val2 = 0
effect_index = 0
last_position = 0
button_state = False
pixel.fill(colors[effect_index])
```

The Loop

In the loop, the rotary encoder position is tracked. Turning the encoder changes the `effect_index` value. This value changes the NeoPixel color and what MIDI CC message type is sent.

```
while True:
    position = -encoder.position
    if position != last_position:
        # encoder changes midi CC message type
        if position > last_position:
            effect_index = (effect_index + 1) % len(messages)
        else:
            effect_index = (effect_index - 1) % len(messages)
        pixel.fill(colors[effect_index])
        last_position = position
```

Panic Button

If you press the button the rotary encoder, then the MIDI panic command is sent. That command turns off all notes. It also sends a sustain command that turns sustain off. When the button is pressed, the NeoPixel turns white.

```
if not button.value and not button_state:
    pixel.fill((255, 255, 255))
    # button press sends MIDI panic
    panic = ControlChange(123, 0)
    midi.send(panic)
    # and turns sustain off
    sus_off = ControlChange(64, 0)
    midi.send(sus_off)
    button_state = True
if button.value and button_state:
    # reset neopixel to CC msg color
    pixel.fill(colors[effect_index])
    button_state = False
```

Under Pressure

When a pressure reading is ready from the BMP585, it's checked against the previous pressure reading. If the difference between the two values is greater than the `change_sense` value, the reading is mapped to a MIDI CC value between `0` and `127`.

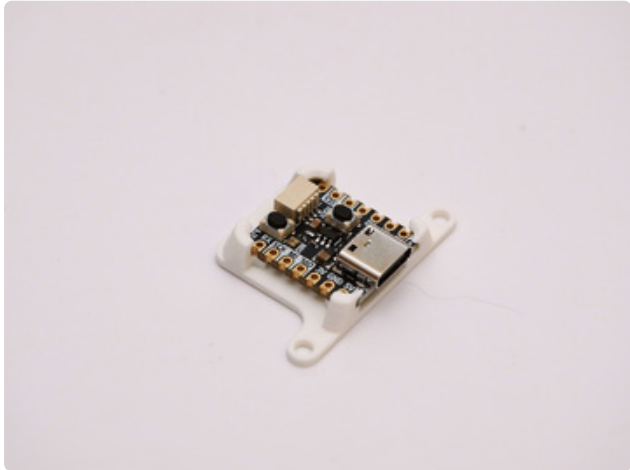
The mapped value is sent as part of the MIDI CC message. The message type is determined by the value of `effect_index`, which is used as the index value in the `messages` list. The following CC message types are in the list:

1. Modulation (1)
2. Breath Controller (2)
3. Volume (7)
4. Sustain (64)
5. ChannelPressure

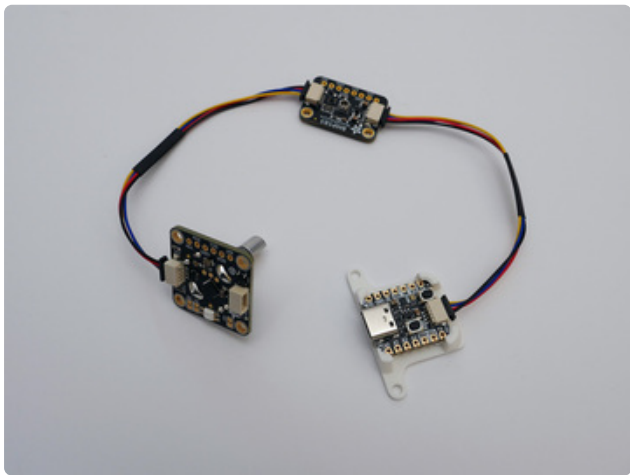
You can add additionally message types to the messages list to use. In testing, these felt like the most effective to pair with breath control.

```
if bmp.data_ready:
    # get pressure reading
    pressure = bmp.pressure
    # if the pressure has changed enough
    # (adjust change_sense value at top to inc or dec)
    if abs(pressure - mod_val2) > change_sense:
        # map pressure reading to CC range
        mod_val1 = round(simpleio.map_range(pressure, low_press, high_press, 0,
127))
        # updates previous value to hold current value
        mod_val2 = pressure
        # MIDI data has to be sent as an integer
        modulation = int(mod_val1)
        # possible midi messages determined by effect_index value:
        # 1: modulation
        # 2: breath controller
        # 7: volume
        # 64: sustain
        # ChannelPressure(modulation)
        if effect_index < 4:
            # prep CC message with CC number and value as mapped pressure
            modWheel = ControlChange(messages[effect_index], modulation)
        else:
            # prep Channel Pressure message with value as mapped pressure
            modWheel = ChannelPressure(modulation)
        # CC message is sent
        midi.send(modWheel)
        # print(modWheel)
        # delay to settle MIDI data
        time.sleep(0.001)
```

Assembly



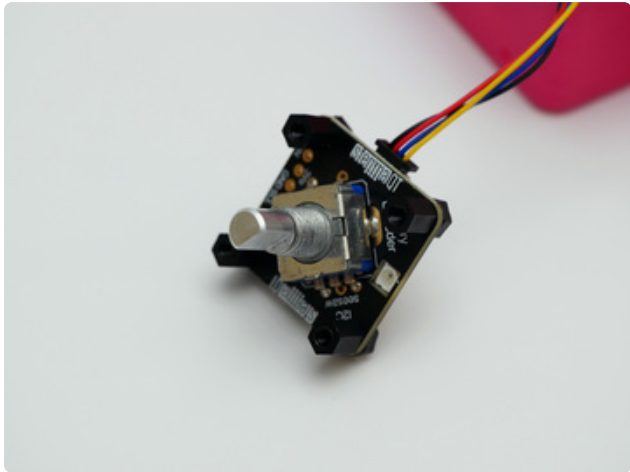
Insert the QT Py into the mount by flexing the back of the mount.



Connect the QT Py, BMP585 and rotary encoder together with STEMMA QT cables.



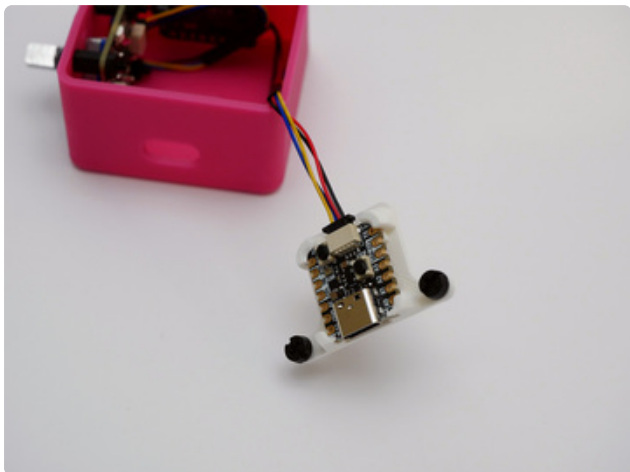
Start by mounting the BMP585 to the front of the case with M2.5 screws and nuts.



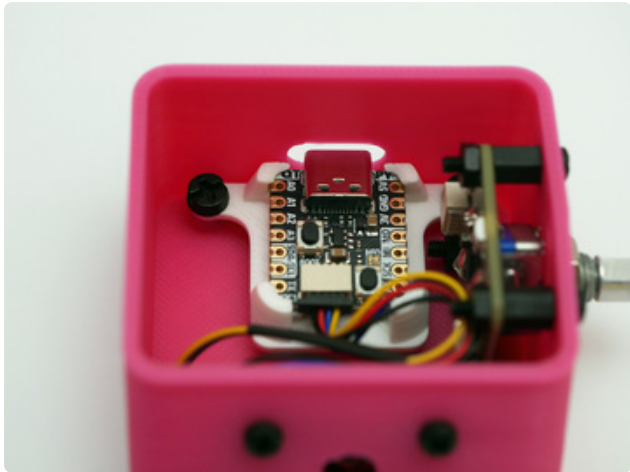
Attach M2.5 stand-offs and nuts to the rotary encoder.



Attach the rotary encoder to the enclosure with M2.5 screws for the stand-offs and the nut for the rotary encoder.



Insert M3 screws into the two mounting holes on the QT Py mount.



Secure the QT Py to the M3 mounting holes inside the enclosure.



Insert the NeoPixel diffuser into the cutout on the case.



Put the lid on top of the case. It has a track that the case edge pops into.

You can add a knob to the rotary encoder to use it a bit easier.



Push the tubing into the cutout for the BMP585 sensor.



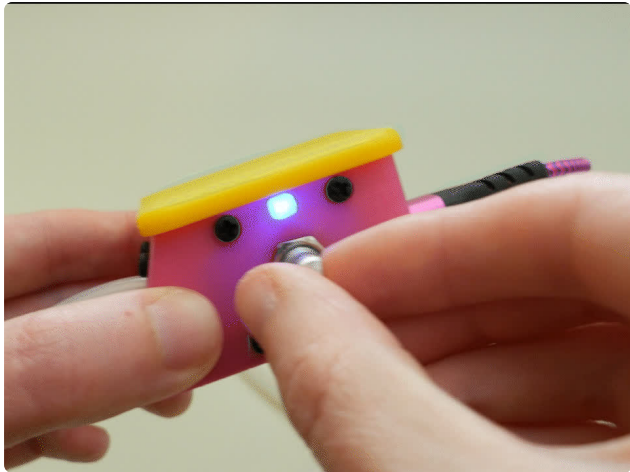
Insert the other end of the tube into the mouthpiece.



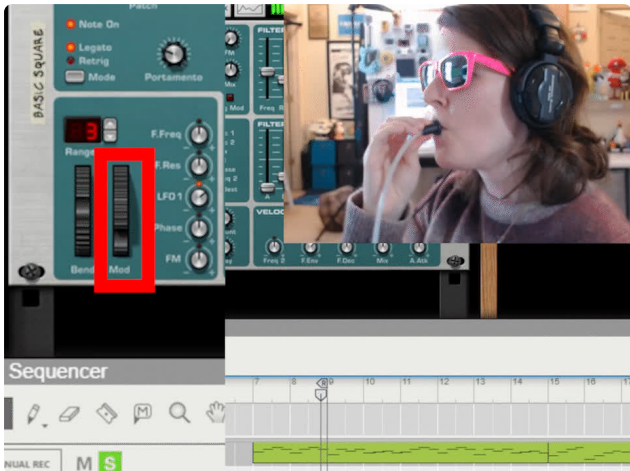
That completes the assembly!

Use

Connect the controller to your computer with a USB C cable. Then, open up your DAW or other software that takes in MIDI.



Use the rotary encoder to select the MIDI CC message that you want to send.



After selecting your MIDI CC message, you can start controlling your virtual synthesizers with air.

Customize the Code

You can change the pressure sensitivity range from the BMP585 sensor at the top of the code:

```
low_press = 970
high_press = 1012
```

You can change the MIDI CC message and the associated NeoPixel color by editing the `messages` and the `colors` arrays right before the loop.

```
# CC message channels
# last index is place holder for Channel Pressure message type
messages = [1, 2, 7, 64, 0]
# neopixel colors to associate with CC messages
colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255),
          (255, 255, 0), (255, 0, 255)]
```