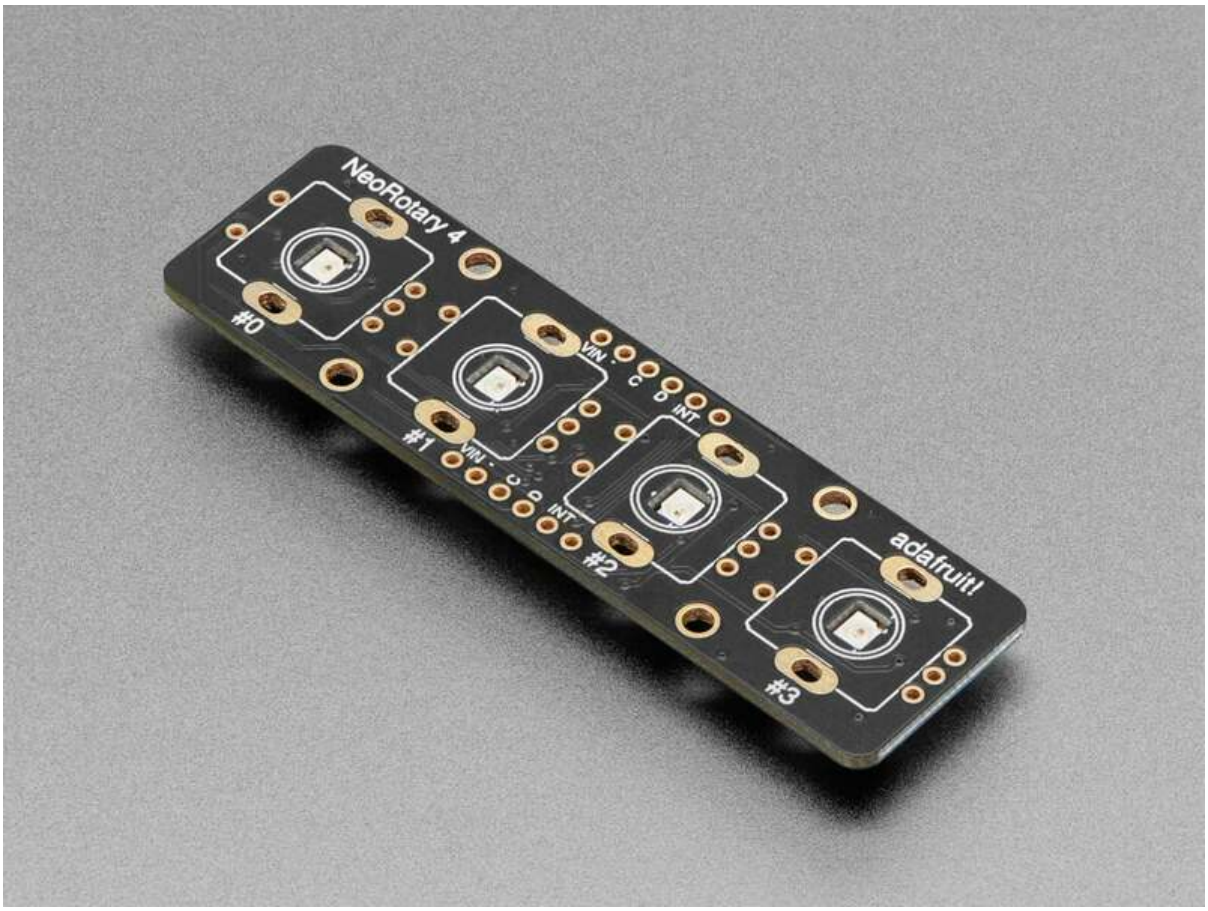
Adafruit I2C Quad Rotary Encoder Breakout

Created by Liz Clark

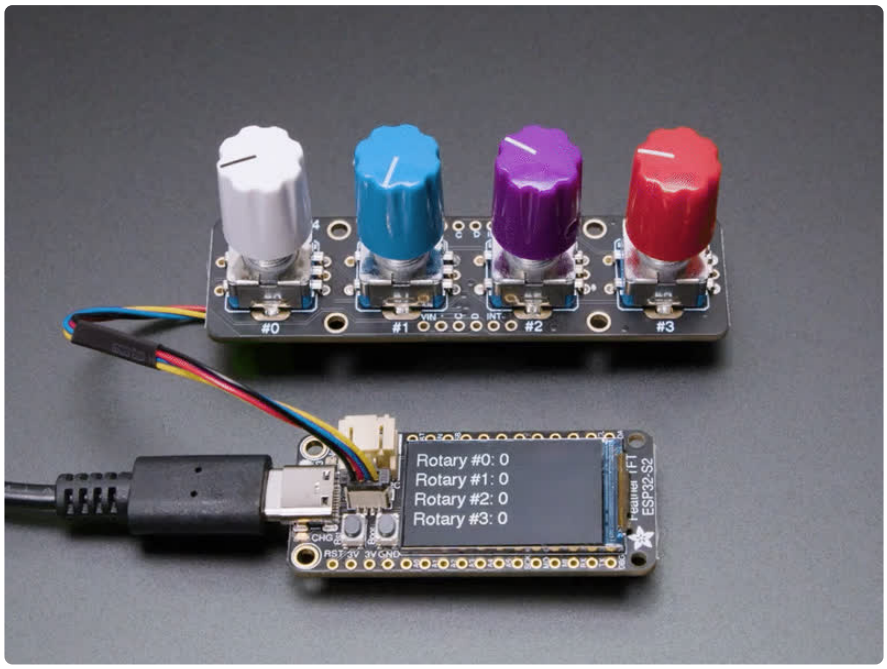


https://learn.adafruit.com/adafruit-i2c-quad-rotary-encoder-breakout

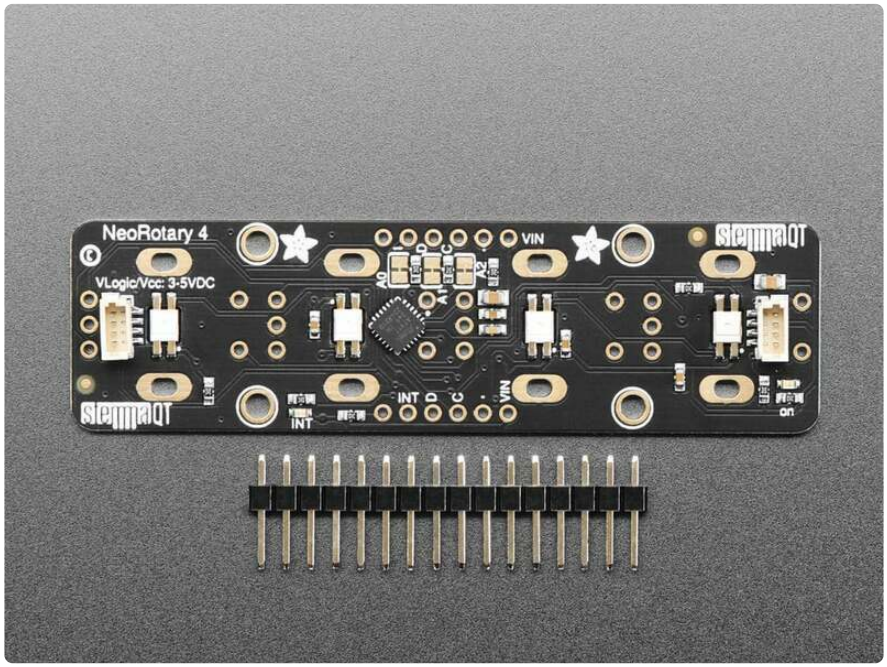
Last updated on 2024-02-26 01:40:28 PM EST

Table of Contents

Overview



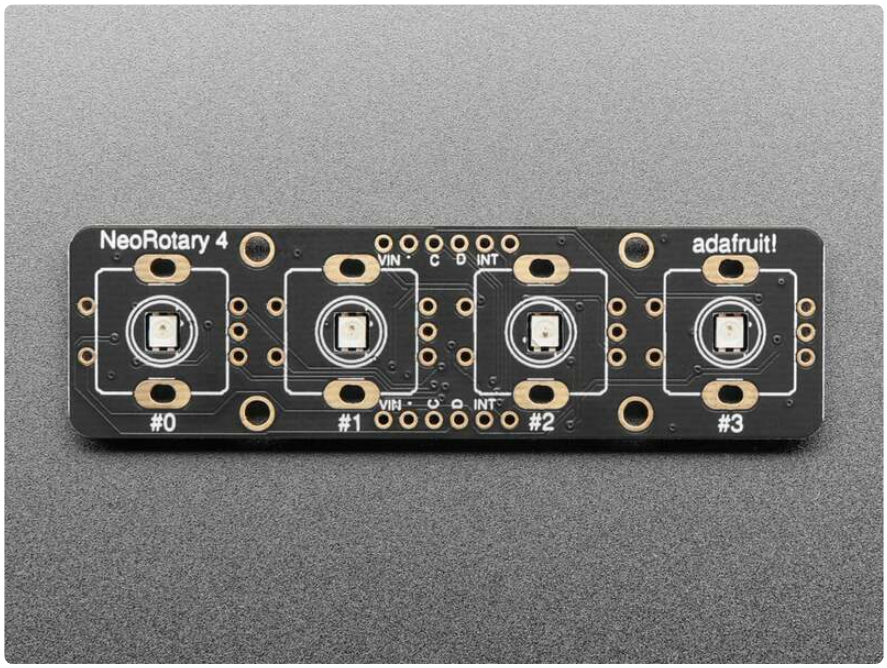
Rotary encoders are soooo much fun! Twist 'em this way, then twist them that way. Unlike potentiometers, they go all the way around and often have little detents for tactile feedback. But, if you've ever tried to add encoders to your project you know that they're a real challenge to use: timers, interrupts, debouncing...



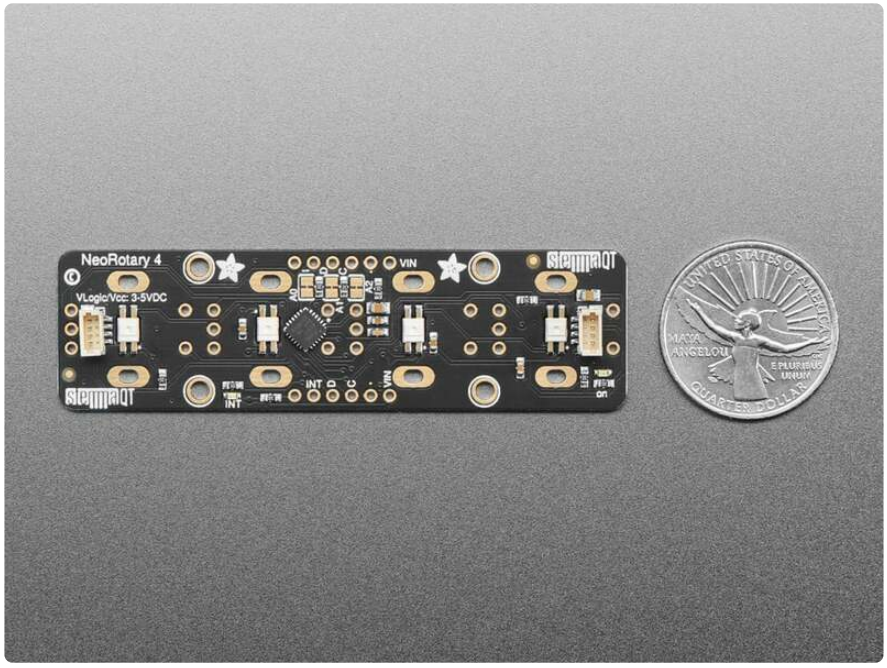
This Stemma QT breakout makes all that frustration go away - and allows you to read up to 4 encoders for big builds with lots of twisty interfaces. You can solder in any four 'standard' PEC11-pinout rotary encoders with or without a push-switch. The onboard microcontroller is programmed with our seesaw firmware and will track all

pulses and pins for you and then save the incremental value for querying at any time over I2C. Plug it in with a Stemma QT cable for instant rotary goodness, with any kind of microcontroller from an Arduino UNO up to a Raspberry Pi.

You can use our Arduino library to control and read data (https://adafru.it/BrV) with any compatible microcontroller. We also have CircuitPython/Python code (https://adafru.it/BrW) for use with computers or single-board Linux boards.

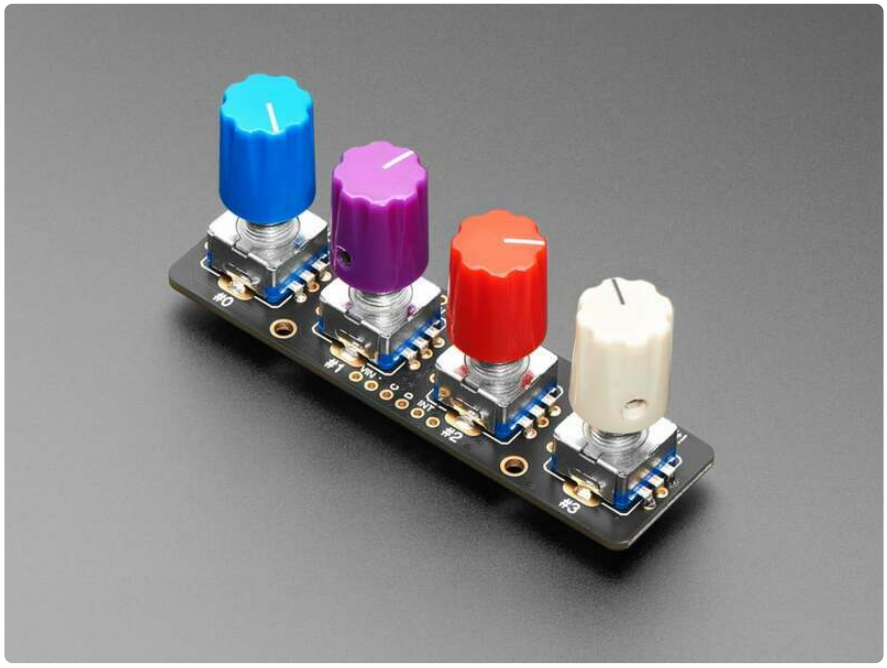


It's also easy to add this breakout to a breadboard - with six 0.1"-spaced breakout pads. Power with 3 to 5V DC and then use 3 or 5V logic I2C data. The INT pin can be configured to pulse low whenever rotation or push-buttoning is detected so you do not have to spam-read the I2C port to detect motion.

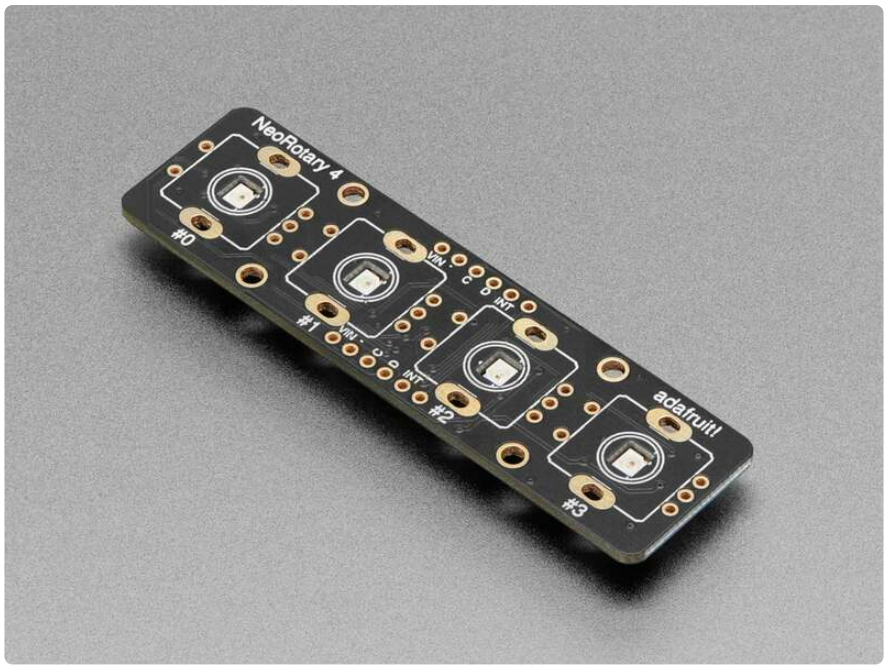


If you happen to be using clear/translucent shaft encoders, there are reverse-mount NeoPixels on board, that can display any color you like, they are controlled over I2C for additional visual feedback or keep them off if you like. Note that for metal-shaft encoders, the LEDs are not visible. On the back, there's a green power LED as well as a red INT LED that, if the interrupt is configured, will blink when the interrupt fires.

Using the three onboard address jumpers, you can connect up to 8 of these encoders on a single I2C port. The first one will be at address 0x49, the last one at 0x50 when all three jumpers are cut open.

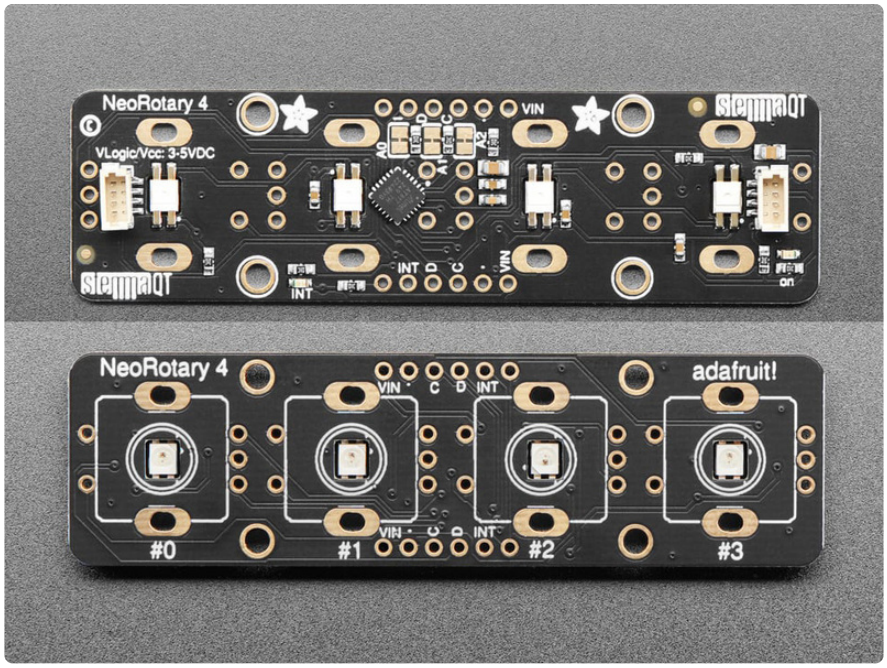


To get you going fast, we spun up a custom-made PCB with the seesaw chip and all supporting circuitry, in the **STEMMA QT** form factor (https://adafru.it/LBQ), making them easy to interface with. The STEMMA QT connectors (https://adafru.it/JqB) on either side are compatible with the SparkFun Qwiic (https://adafru.it/Fpw) I2C connectors. This allows you to make solderless connections between your development board and the rotary encoder or to chain them with a wide range of other sensors and accessories using a **compatible cable** (https://adafru.it/JnB). A QT Cable is not included, but we have a variety in the shop (https://adafru.it/17VE).



This breakout does not come with any encoders soldered on, so you can pick whatever encoder you like. We sell a common 24-detent-with-switch encoder here and it works wonderfully. (http://adafru.it/377) You can also use encoders without detents or with a different number of detents per rotation, of course! You'll need to solder the encoders and optional header onto the PCB to use with a solderless breadboard. but it's fairly easy and takes only a few minutes even for a beginner.

Pinouts



The default I2C address is **0x49**.

Power Pins

- **VIN** - This is the power pin. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 3V microcontroller like a Feather RP2040, use 3V, or for a 5V microcontroller like Arduino, use 5V.
- **- (GND)** - This is common ground for power and logic.

I2C Logic Pins

The default I2C address is **0x49**.

- **C** - I2C clock pin (**SCL**), connect to your microcontroller I2C clock line. There's a **10K pullup** on this pin.
- **D** - I2C data pin (**SDA**), connect to your microcontroller I2C data line. There's a **10K pullup** on this pin.
- **STEMMA QT** (https://adafru.it/Ft4) **-** These connectors allow you to connect to development boards with **STEMMA QT** / Qwiic connectors or to other things with various associated accessories (https://adafru.it/JRA).

Address Jumpers

On the back of the board are **three address jumpers**, labeled **A0**, **A1**, and **A2**, above the breakout pads along the bottom of the board. These jumpers allow you to chain up to 8 of these boards on the same pair of I2C clock and data pins. To do so, you cut the jumpers "open" by separating the two pads.

If you happen to need more than 8, it's possible to set the I2C address with a special address-change command that is saved to the onboard non-volatile EEPROM memory.

The default I2C address is **0x49**. The other address options can be calculated by "adding" the **A0/A1/A2** to the base of **0x49**.

A0 sets the lowest bit with a value of **1**, **A1** sets the next bit with a value of **2** and **A2** sets the next bit with a value of **4**. The final address is **0x49** + **A2** + **A1** + **A0** which would be **0x50**.

If only **A0** is cut, the address is **0x49** + **1** = **0x49**

If only **A1** is cut, the address is **0x49** + **2** = **0x4B**

If only **A2** is cut, the address is **0x49** + **4** = **0x4D**

The table below shows all possible addresses, and whether the pin(s) should be **L**ow (left closed) or **H**igh (cut open).

ADDR	A0	A1	A2
0x49	L	L	L
0x4A	H	L	L
0x4B	L	H	L
0x4C	H	H	L
0x4D	L	L	H
0x4E	H	L	H
0x4F	L	H	H
0x50	H	H	H

Interrupt Pin and LED

- **INT** - This is the interrupt output pin. It can be configured to pulse low whenever rotation or push-buttoning is detected so you do not have to spam-read the I2C port to detect motion.
- **Interrupt LED** - On the back of the board to the left of the bottom row of breakout pins is the interrupt LED. It is the red LED and turns on whenever an interrupt is detected.

UPDI Pin

The UPDI pin is not labeled on the board silk and is located next to the interrupt (**INT**) pin.

- **UPDI** - This is the single-pin **U**nified **P**rogram and **D**ebug **I**nterface. This pin is for external programming or on-chip-debugging for the ATtiny817 running the seesaw firmware (https://adafru.it/VdL). We have a page in the ATtiny Breakouts with seesaw Learn Guide (https://adafru.it/18ED) detailing how to reprogram these chips with your own firmware (at your own risk). We don't provide any support for custom builds of seesaw - we think this is cool and useful for the Maker community.

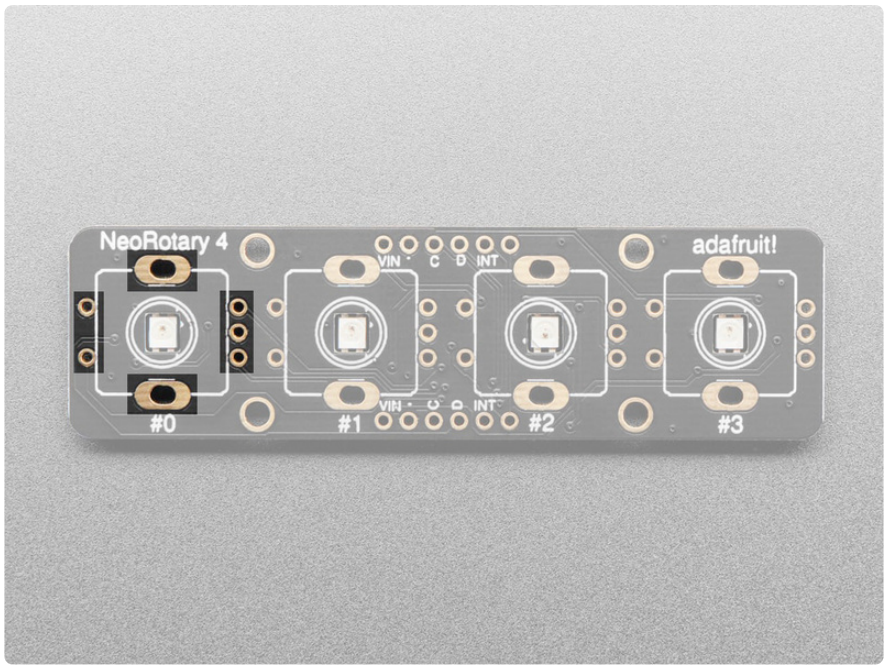
Power LED

- **Power LED -** On the back of the board, below the STEMMA connector on the right, is the power LED, labeled **on**. It is the green LED.

Rotary Encoder Pins

On the front of the board are outlines for four rotary encoders on the board silk. They are labeled **#0** to **#3**. This lets you know where you should place the rotary encoders for soldering. **This breakout does not come with any encoders soldered on.** We sell a common 24-detent-with-switch encoder here and it works wonderfully. (http://adafru.it/377)

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



Each of the four encoder sections has pins for reading the encoder and a switch. The encoder pins are the three pins on the right side of the encoder outline. The encoder A pin is towards the bottom of the board and the encoder B pin is towards the top of the board. The center pin is ground.

The switch pins are the two pins on the left side of the encoder outline. The switch pin is towards the bottom of the board and the pin towards the top of the board is ground. The two larger oval pins are both connected to ground.

These are **the pin names in the seesaw firmware** for each rotary encoder:

- **Encoder #0**
  - Switch: pin **12**
  - Encoder A: pin **8**
  - Encoder B: pin **9**
- **Encoder #1**
  - Switch: pin **14**
  - Encoder A: pin **10**
  - Encoder B: pin **11**
- **Encoder #2**
  - Switch: pin **17**
  - Encoder A: pin **2**
  - Encoder B: pin **3**

- **Encoder #3**
    - Switch: pin **9**
    - Encoder A: pin **4**
    - Encoder B: pin **5**

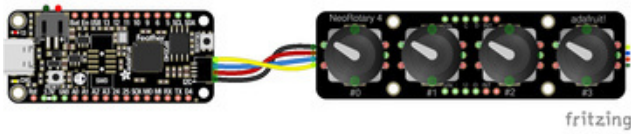
CircuitPython & Python

It's easy to use the **I2C Quad Rotary Encoder breakout** with Python or CircuitPython, and the Adafruit_CircuitPython_seesaw (https://adafru.it/BrW) module. This module allows you to easily write Python code that reads each encoder position (relative to the starting position) and the four button presses on each encoder.

You can use this adapter with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

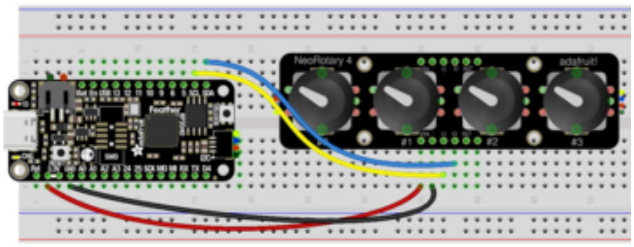
CircuitPython Microcontroller Wiring

First wire up an I2C adapter to your board exactly as follows. The following is the adapter wired to a Feather RP2040 using the STEMMA connector:



Board STEMMA 3V to **breakout VIN (red wire)**
Board STEMMA GND to **breakout GND (black wire)**
Board STEMMA SCL to **breakout SCL (yellow wire)**
Board STEMMA SDA to **breakout SDA (blue wire)**

The following is the adapter wired to a Feather RP2040 using a solderless breadboard:

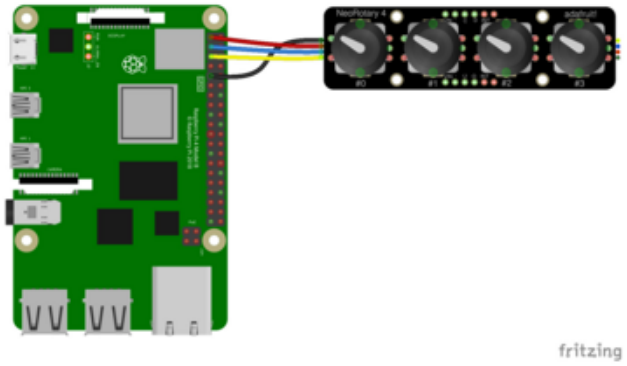


Board 3V to **breakout VIN (red wire)**
Board GND to **breakout GND (black wire)**
Board SCL to **breakout SCL (yellow wire)**
Board SDA to **breakout SDA (blue wire)**

Python Computer Wiring

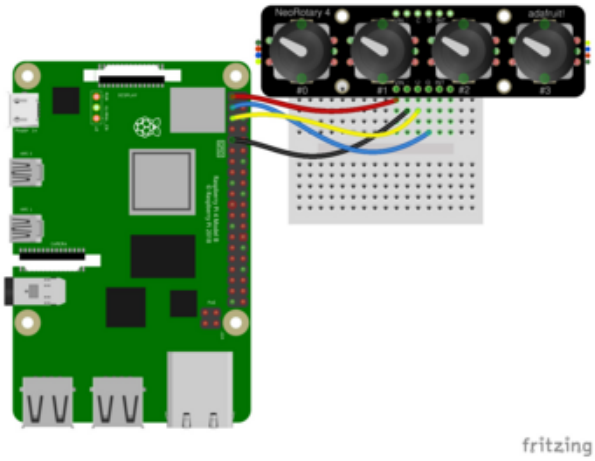
Since there are dozens of Linux computers/boards you can use, we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Here's the Raspberry Pi wired with I2C using the STEMMA connector:



Pi 3V to **breakout VIN (red wire)**
Pi GND to **breakout GND (black wire)**
Pi SCL to **breakout SCL (yellow wire)**
Pi SDA to **breakout SDA (blue wire)**

Here's the Raspberry Pi wired with I2C using a solderless breadboard:



Pi 3V to **breakout VIN (red wire)**
Pi GND to **breakout GND (black wire)**
Pi SCL to **breakout SCL (yellow wire)**
Pi SDA to **breakout SDA (blue wire)**

Python Installation of seesaw Library

You'll need to install the **Adafruit_Blinka** library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

Once that's done, from your command line run the following command:

- `pip3 install adafruit-circuitpython-seesaw`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

CircuitPython Usage

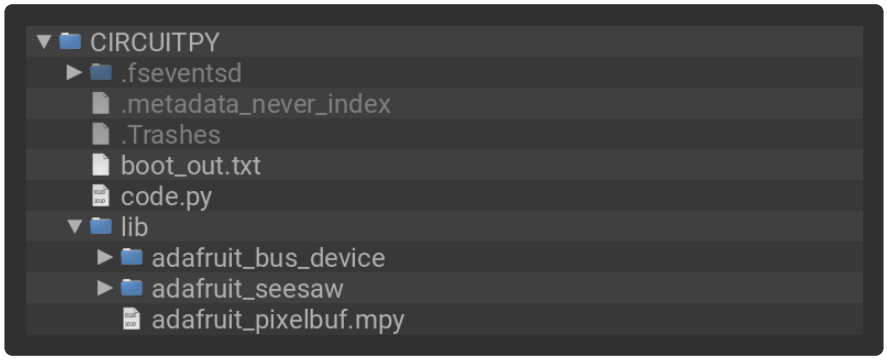
To use with CircuitPython, you need to first install the **Adafruit_CircuitPython_seesaw** library, and its dependencies, into the **lib** folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

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, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

Your **CIRCUITPY/lib** folder should contain the following folders and file:

- **adafruit_bus_device/**

- adafruit_seesaw/
- adafruit_pixelbuf.mpy



Python Usage

Once you have the library `pip3` installed on your computer, copy or download the following example to your computer, and run the following, replacing **code.py** with whatever you named the file:

```
python3 code.py
```

Example Code

If running CircuitPython: Once everything is saved to the **CIRCUITPY** drive, connect to the serial console (https://adafru.it/Bec) to see the data printed out!

If running Python: The console output will appear wherever you are running Python.

```
# SPDX-FileCopyrightText: 2023 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""Quad I2C rotary encoder NeoPixel color picker example."""
import board
from rainbowio import colorwheel
import digitalio
import adafruit_seesaw.seesaw
import adafruit_seesaw.neopixel
import adafruit_seesaw.rotaryio
import adafruit_seesaw.digitalio

# For boards/chips that don't handle clock-stretching well, try running I2C at 50KHz
# import busio
# i2c = busio.I2C(board.SCL, board.SDA, frequency=50000)
# For using the built-in STEMMA QT connector on a microcontroller
i2c = board.STEMMA_I2C()
seesaw = adafruit_seesaw.seesaw.Seesaw(i2c, 0x49)

encoders = [adafruit_seesaw.rotaryio.IncrementalEncoder(seesaw, n) for n in
range(4)]
switches = [adafruit_seesaw.digitalio.DigitalIO(seesaw, pin) for pin in (12, 14, 17,
9)]
```

```
for switch in switches:
    switch.switch_to_input(digitalio.Pull.UP)  # input & pullup!

# four neopixels per PCB
pixels = adafruit_seesaw.neopixel.NeoPixel(seesaw, 18, 4)
pixels.brightness = 0.5

last_positions = [-1, -1, -1, -1]
colors = [0, 0, 0, 0]  # start at red

while True:
    # negate the position to make clockwise rotation positive
    positions = [encoder.position for encoder in encoders]
    print(positions)
    for n, rotary_pos in enumerate(positions):
        if rotary_pos != last_positions[n]:
            if switches[n].value:  # Change the LED color if switch is not pressed
                if (
                    rotary_pos > last_positions[n]
                ):  # Advance forward through the colorwheel.
                    colors[n] += 8
                else:
                    colors[n] -= 8  # Advance backward through the colorwheel.
                colors[n] = (colors[n] + 256) % 256  # wrap around to 0-256
            # Set last position to current position after evaluating
            print(f"Rotary #{n}: {rotary_pos}")
            last_positions[n] = rotary_pos

        # if switch is pressed, light up white, otherwise use the stored color
        if not switches[n].value:
            pixels[n] = 0xFFFFFF
        else:
            pixels[n] = colorwheel(colors[n])
```

In the example, each rotary encoder position is printed to the serial console as it changes. As the encoder position changes, the NeoPixel underneath advances through the rainbow. If you press a rotary encoder button, the NeoPixel underneath turns white.

```
CircuitPython REPL
Auto-reload is on. Simply save files over USB to run them or enter
REPL to disable.
code.py output:
Rotary #0: 0
Rotary #1: 0
Rotary #2: 0
Rotary #3: 0
Rotary #3: 1
Rotary #3: 2
Rotary #2: -1
Rotary #2: -2
Rotary #1: 1
Rotary #1: 2
Rotary #1: 3
```

I2C Clock Stretching

For boards that don't handle clock-stretching well, like Raspberry Pi, you may want to reduce the I2C clock speed to 50KHz by following the directions in this guide. (https://adafru.it/18MF)

Then, at the beginning of the example code, uncomment the **busio** I2C instantiation and comment out the `STEMMA_I2C()` instantiation.

```
# For boards/chips that don't handle clock-stretching well, try running I2C at 50KHz
import busio
i2c = busio.I2C(board.SCL, board.SDA, frequency=50000)
# For using the built-in STEMMA QT connector on a microcontroller
# i2c = board.STEMMA_I2C()
seesaw = adafruit_seesaw.seesaw.Seesaw(i2c, 0x49)
```

For more information on I2C clock stretching and Raspberry Pi, check out this Learn Guide (https://adafru.it/18Na).

Raspberry Pi I2C Clock Stretching Fixes Learn Guide

https://adafru.it/18Nb

Python Docs

Python Docs (https://adafru.it/18EG)

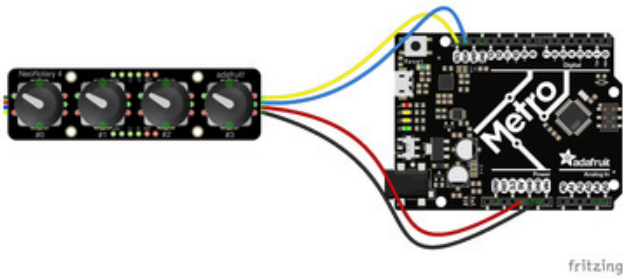
Arduino

Using the I2C quad rotary encoder breakout with Arduino involves wiring up the breakout to your Arduino-compatible microcontroller, installing the Adafruit_Seesaw (https://adafru.it/BrV) library and running the provided example code.

Wiring

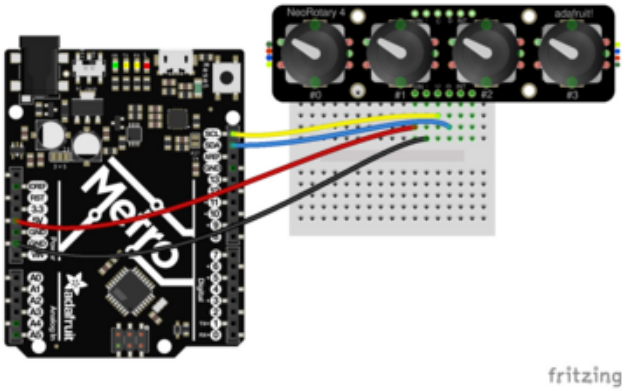
Wire as shown for a **5V** board like an Uno. If you are using a **3V** board, like an Adafruit Feather, wire the board's 3V pin to the breakout VIN.

Here is an Adafruit Metro wired up to the breakout using the STEMMA QT connector:



Board 5V to **breakout VIN (red wire)**
Board GND to **breakout GND (black wire)**
Board SCL to **breakout SCL (yellow wire)**
Board SDA to **breakout SDA (blue wire)**

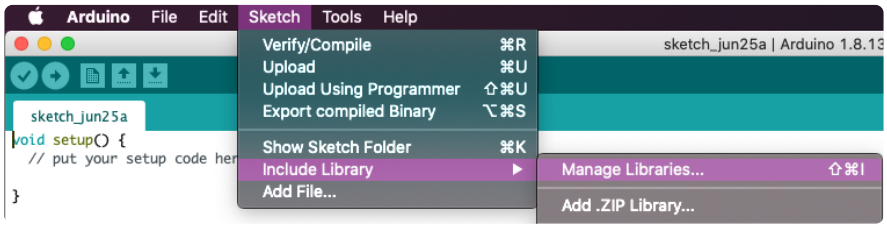
Here is an Adafruit Metro wired up using a solderless breadboard:



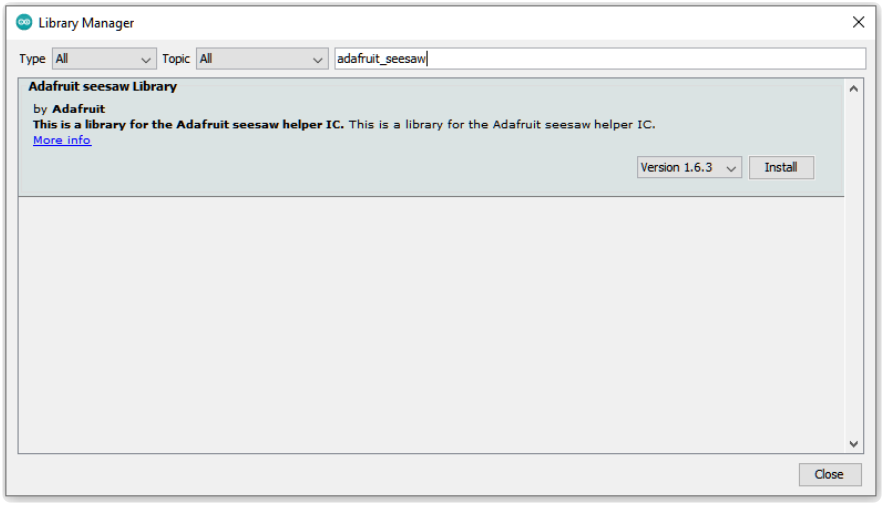
Board 5V to **breakout VIN (red wire)**
Board GND to **breakout GND (black wire)**
Board SCL to **breakout SCL (yellow wire)**
Board SDA to **breakout SDA (blue wire)**

Library Installation

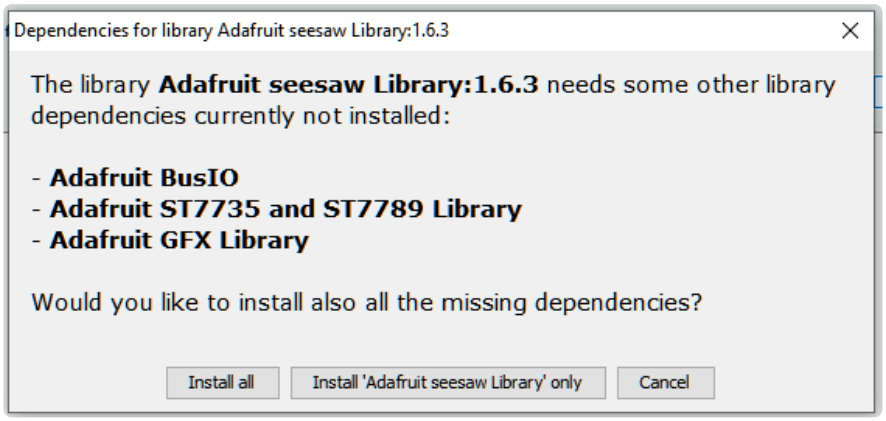
You can install the **Adafruit_Seesaw** library for Arduino using the Library Manager in the Arduino IDE.



Click the **Manage Libraries ...** menu item, search for **Adafruit_Seesaw**, and select the **Adafruit seesaw Library** library:



If asked about dependencies, click "Install all".



If the "Dependencies" window does not come up, then you already have the dependencies installed.

If the dependencies are already installed, you must make sure you update them through the Arduino Library Manager before loading the example!

Example Code

```
/*
 * This is a demo for a QT Py RP2040 connected to a quad rotary encoder breakout
 * using the onboard Stemma QT Port
 * https://www.adafruit.com/product/4900
 * https://www.adafruit.com/product/5752
 *
 */
#include "Adafruit_seesaw.h"
#include <seesaw_neopixel.h>

#define SS_NEO_PIN       18
#define SS_ENC0_SWITCH   12
#define SS_ENC1_SWITCH   14
#define SS_ENC2_SWITCH   17
#define SS_ENC3_SWITCH   9
```

```
#define SEESAW_ADDR      0x49

Adafruit_seesaw ss = Adafruit_seesaw(&Wire);
seesaw_NeoPixel pixels = seesaw_NeoPixel(4, SS_NEO_PIN, NEO_GRB + NEO_KHZ800);

int32_t enc_positions[4] = {0, 0, 0, 0};

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);

  Serial.println("Looking for seesaw!");

  if (! ss.begin(SEESAW_ADDR) || !pixels.begin(SEESAW_ADDR)) {
    Serial.println("Couldn't find seesaw on default address");
    while(1) delay(10);
  }
  Serial.println("seesaw started");
  uint32_t version = ((ss.getVersion() >> 16) & 0xFFFF);
  if (version  != 5752){
    Serial.print("Wrong firmware loaded? ");
    Serial.println(version);
    while(1) delay(10);
  }
  Serial.println("Found Product 5752");

  ss.pinMode(SS_ENC0_SWITCH, INPUT_PULLUP);
  ss.pinMode(SS_ENC1_SWITCH, INPUT_PULLUP);
  ss.pinMode(SS_ENC2_SWITCH, INPUT_PULLUP);
  ss.pinMode(SS_ENC3_SWITCH, INPUT_PULLUP);
  ss.setGPIOInterrupts(1UL << SS_ENC0_SWITCH | 1UL << SS_ENC1_SWITCH |
                       1UL << SS_ENC2_SWITCH | 1UL << SS_ENC3_SWITCH, 1);


  // get starting positions
  for (int e=0; e<4; e++) {
    enc_positions[e] = ss.getEncoderPosition(e);
    ss.enableEncoderInterrupt(e);
  }

  Serial.println("Turning on interrupts");

  pixels.setBrightness(255);
  pixels.show(); // Initialize all pixels to 'off'
}

void loop() {

  if (! ss.digitalRead(SS_ENC0_SWITCH)) {
    Serial.println("ENC0 pressed!");
  }
  if (! ss.digitalRead(SS_ENC1_SWITCH)) {
    Serial.println("ENC1 pressed!");
  }
  if (! ss.digitalRead(SS_ENC2_SWITCH)) {
    Serial.println("ENC2 pressed!");
  }
  if (! ss.digitalRead(SS_ENC3_SWITCH)) {
    Serial.println("ENC3 pressed!");
  }


  for (int e=0; e<4; e++) {
    int32_t new_enc_position = ss.getEncoderPosition(e);
    // did we move around?
    if (enc_positions[e] != new_enc_position) {
      Serial.print("Encoder #");
      Serial.print(e);
```

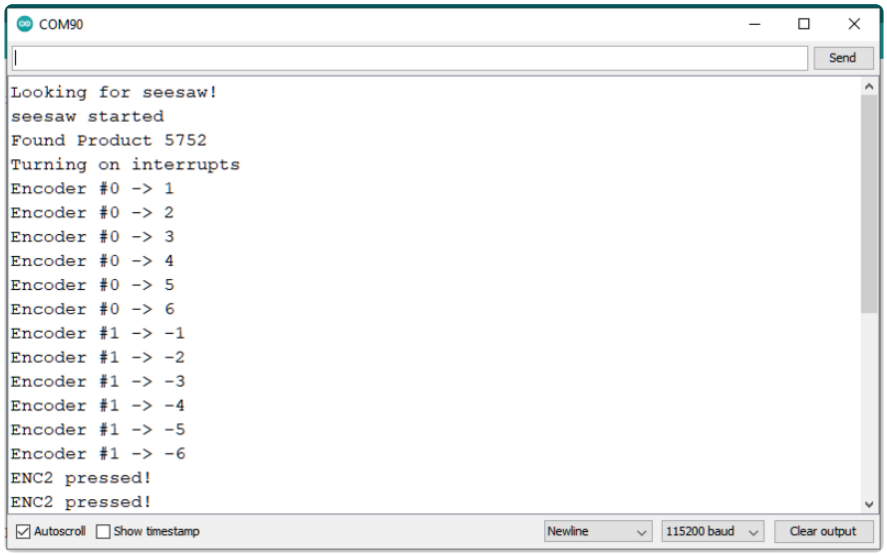
```
      Serial.print(" -> ");
      Serial.println(new_enc_position);      // display new position
      enc_positions[e] = new_enc_position;      // and save for next round

      // change the neopixel color, mulitply the new positiion by 4 to speed it up
      pixels.setPixelColor(e, Wheel((new_enc_position*4) & 0xFF));
      pixels.show();
    }
  }

  // don't overwhelm serial port
  delay(10);
}


uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if (WheelPos < 85) {
    return seesaw_NeoPixel::Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  if (WheelPos < 170) {
    WheelPos -= 85;
    return seesaw_NeoPixel::Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return seesaw_NeoPixel::Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}
```

Upload the sketch to your board and open up the Serial Monitor (**Tools -> Serial Monitor**) at 115200 baud. You'll see the seesaw firmware recognized by the code. Then, when you turn any of the four encoders or press any of the buttons on the encoders it will print to the Serial Monitor. As you turn an encoder, the NeoPixel underneath the encoder will begin to advance through the color rainbow. You'll also see the interrupt LED light up with each encoder turn and button press.



Arduino Docs

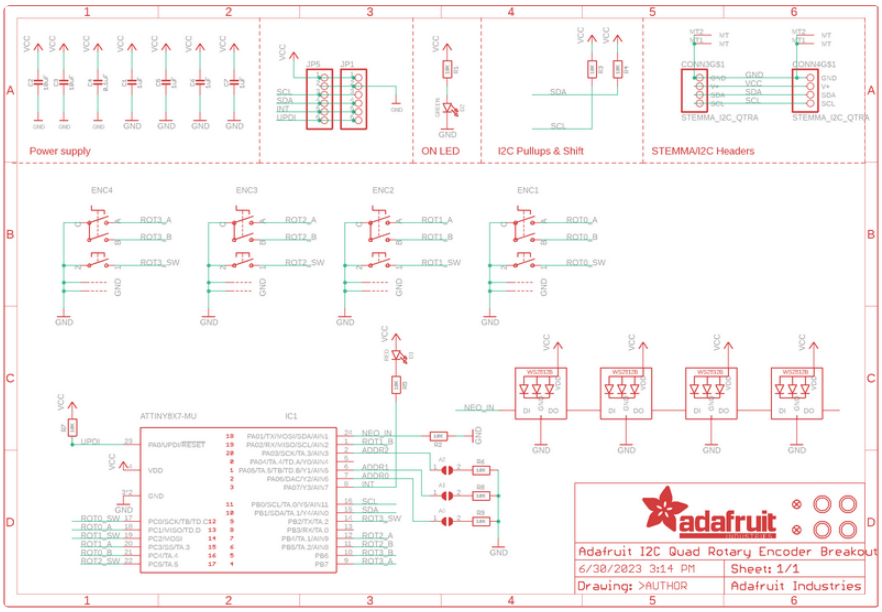
Arduino Docs (https://adafru.it/SdQ)

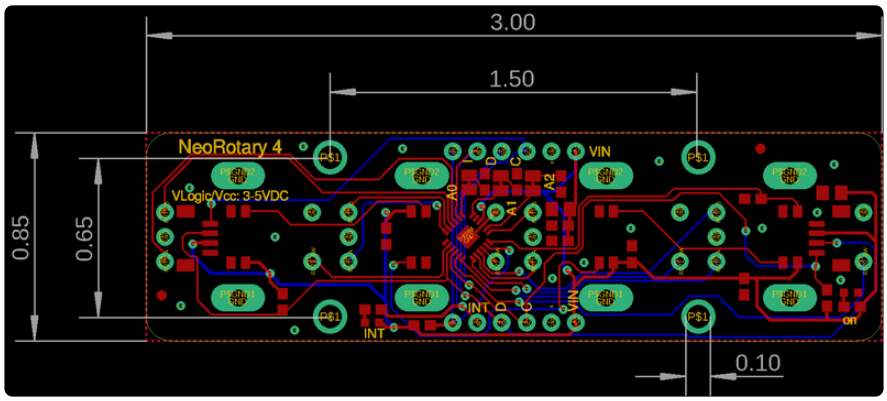
Downloads

Files

- ATtiny817 Datasheet (https://adafru.it/VhF)
- EagleCAD PCB Files on GitHub (https://adafru.it/18Nc)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/18Nd)

Schematic and Fab Print





Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Adafruit:

5752