NAU7802 Pet Food Scale

Created by Liz Clark

https://learn.adafruit.com/nau7802-pet-food-scale

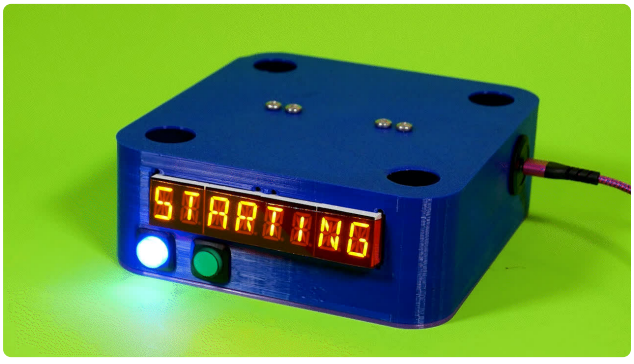
Last updated on 2024-06-03 03:39:34 PM EDT

Table of Contents

Overview

This project uses a QT Py ESP32-S2 running CircuitPython with a NAU7802 to build a food scale that's the perfect size for holding your pet's dry food. A strain gauge is used to measure the weight of a food container on top of the 3D printed platform. It makes for a fun way to see your pet's food consumption over time.

Alphanumeric displays show the current weight in either ounces or grams. They also act as a GUI when calibrating the scale.



Two LED buttons let you interact with the code and help you visualize what is happening during calibration.

This project is not meant to be a perfectly precise scale, but it does do a good job at trend-tracking and letting you know when you're getting low on kibble

Accuracy

By calibrating the NAU7802 properly, you can get some accurate measurements for the items that you're trying to weigh. If your project needs extremely accurate measurements for weight though, then this method might not be a good fit since your readings may be within a small margin of error.

Prerequisite Guides

Adafruit QT Py ESP32-S2

https://adafru.it/10bJ

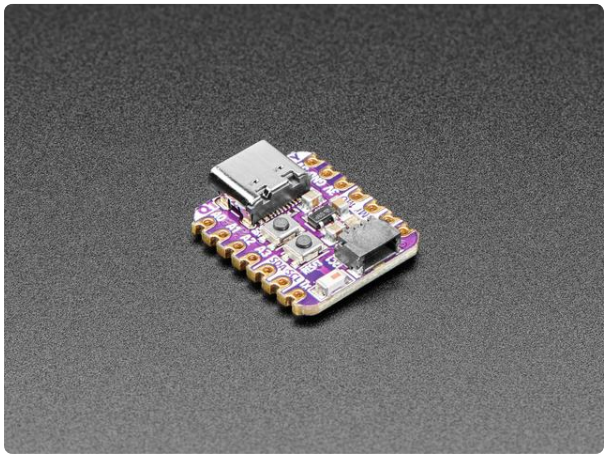
Adafruit NAU7802 24-Bit ADC - STEMMA QT / Qwiic

https://adafru.it/10bK

Adafruit LED Backpacks

https://adafru.it/dAo

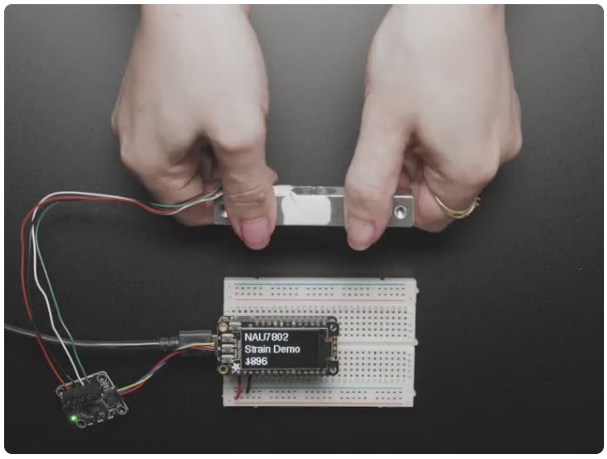
Parts



Adafruit QT Py ESP32-S2 WiFi Dev Board with STEMMA QT

What has your favorite Espressif WiFi microcontroller, comes with our favorite connector - the STEMMA QT, a chainable I2C port, and has...

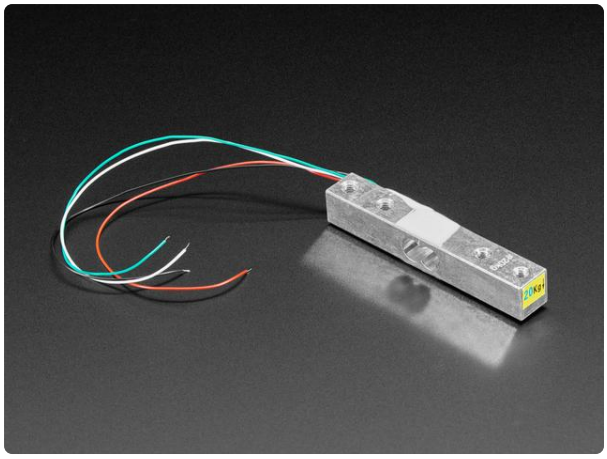
https://www.adafruit.com/product/5325



[Adafruit NAU7802 24-Bit ADC - STEMMA QT / Qwiic]

If you are feeling the stress and strain of modern life a Wheatstone bridge and you want to quantify it, this handy breakout will do the job, no sweat! The Adafruit...

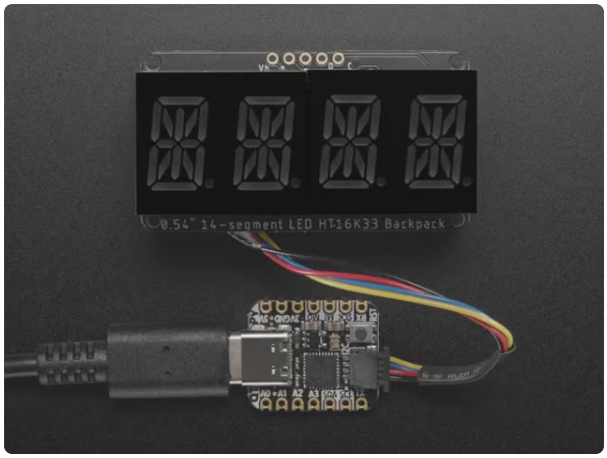
https://www.adafruit.com/product/4538



[Strain Gauge Load Cell - 4 Wires - 20Kg]

A strain gauge is a type of electronic sensor used to measure force or strain (big surprise there). They are made of an insulating flexible backing with a metallic...

https://www.adafruit.com/product/4543



[Quad Alphanumeric Display - Yellow 0.54" Digits w/ I2C Backpack]

Display, elegantly, 012345678 or 9! Gaze, hypnotized, at ABCDEFGHIJKLM - well it can display the whole alphabet. You get the point. This is a nice, bright alphanumeric display that...

https://www.adafruit.com/product/2158

[16mm Illuminated Pushbutton - Green Momentary]

A button is a button, and an LED is a LED, but this LED illuminated button is a lovely combination of both! It's a medium sized button, large enough to press easily but not too big...

https://www.adafruit.com/product/1440

16mm Illuminated Pushbutton - Blue Momentary

A switch is a switch, and an LED is an LED, but this LED illuminated button is a lovely combination of both! It's a medium sized button, large enough to press easily but not too...

https://www.adafruit.com/product/1477

1 x USB C Round Panel Mount Extension Cable — https://www.adafruit.com/product/4218

USB C extension

---

1 x Pink and Purple Woven USB A to USB C Cable - 1 meter long — https://www.adafruit.com/product/5153

USB C to USB A

---

3 x STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long — https://www.adafruit.com/product/4210

STEMMA QT cable

---

1 x Silicone Cover Stranded-Core Wire - 30AWG in Various Colors — https://www.adafruit.com/product/2051

Stranded-Core Wire

---

1 x M4 Screws — https://www.amazon.com/gp/product/B08HQF696N/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1

M4 screws

---

1 x 45mm M4 screws — https://www.lowes.com/pd/Hillman-4mm-0-7-x-45mm-Phillips-Slotted-Combination-Drive-Machine-Screws-30-Count/999994820

45mm M4 screws

---

1 x Food storage container — https://www.target.com/p/oxo-pop-2-6qt-airtight-food-storage-container/-/A-15420440#lnk=sametab

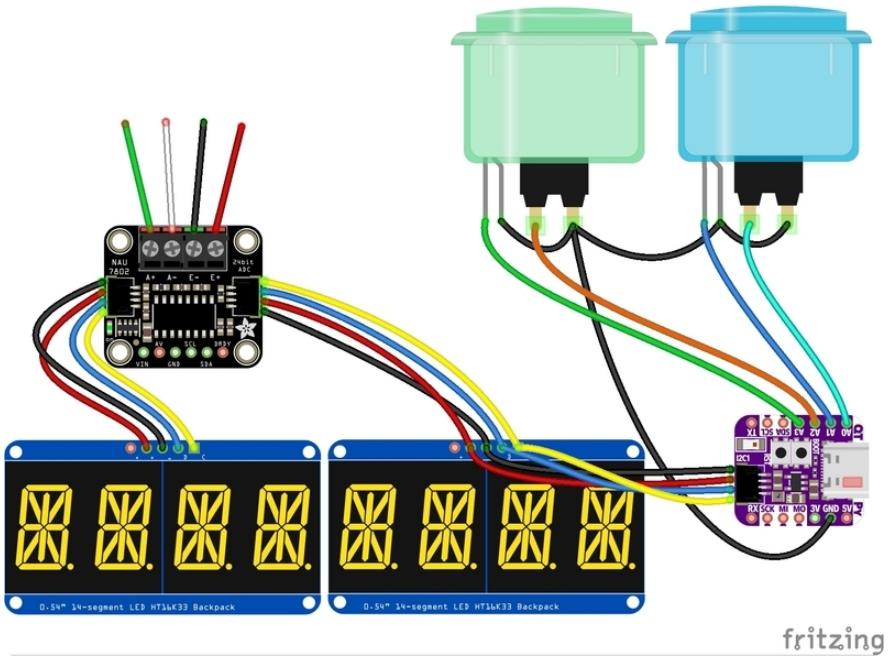
2.6 quart storage container

---

1 x Calibration Weight 1g 2g 5g 10g 20g — https://www.amazon.com/dp/B078Q3JZY7?psc=1&ref=ppx_yo2ov_dt_b_product_details

Calibration weights

Circuit Diagram



Blue Button

- **Button input** to **board A0**
- **Button GND** to **board GND**
- **Button LED** to **board A1**
- **Button LED GND** to **board GND**

Green Button

- **Button input** to **board A2**
- **Button GND** to **board GND**
- **Button LED** to **board A3**
- **Button LED GND** to **board GND**

NAU7802

- **Sensor A+** to **strain gauge green wire**
- **Sensor A-** to **strain gauge white wire**
- **Sensor E-** to **strain gauge black wire**
- **Sensor E+** to **strain gauge red wire**

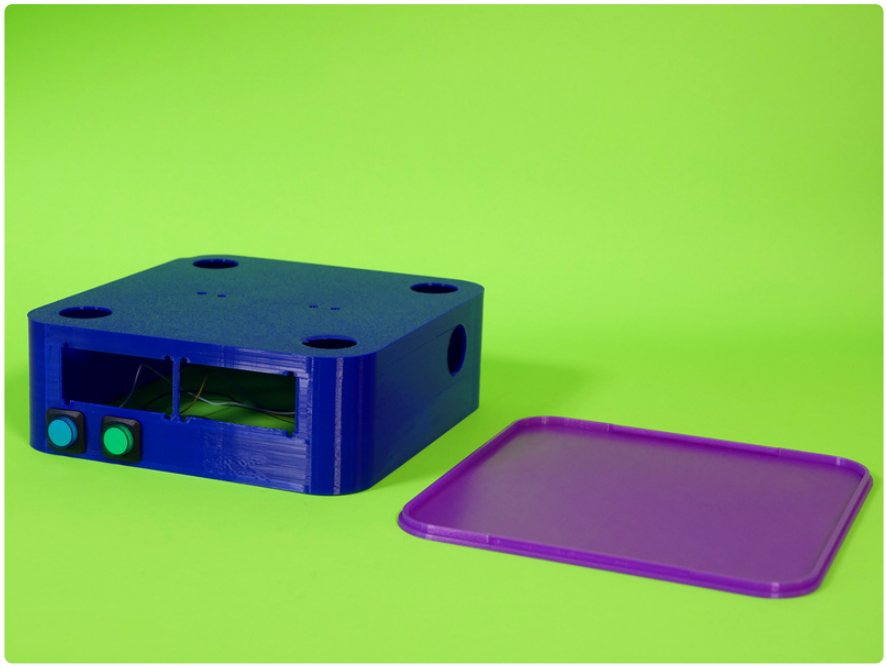
Alphanumeric Display I2C Jumpers

- **Alphanumeric display 1** (on the left): **A0, A1** and **A2 open (0x70)**
- **Alphanumeric display 2** (on the right): **A0 closed, A1** and **A2 open (0x71)**

STEMMA Connections

- Connect alphanumeric display 1, alphanumeric display 2 and the NAU7802 together with STEMMA QT cables. Plug the STEMMA QT cable into the QT Py ESP32-S2

3D Printing



The food scale may be assembled in a 3D printed case, with 3D printed stand-offs for the strain gauge, described below. The top of the case benefits from some supports.

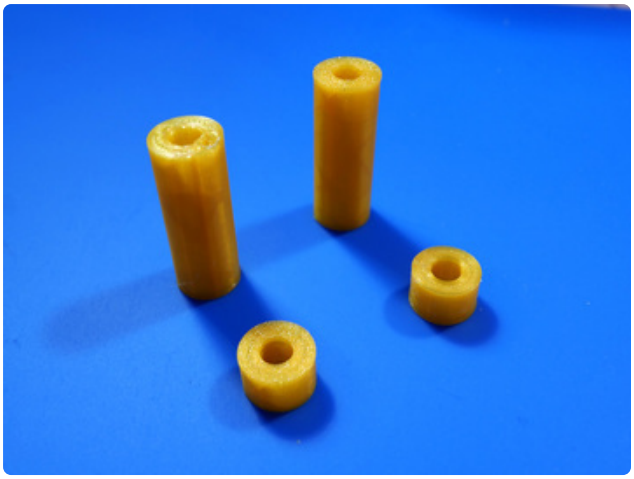
The STL files can be downloaded directly here or from Thingiverse.

Thingiverse download

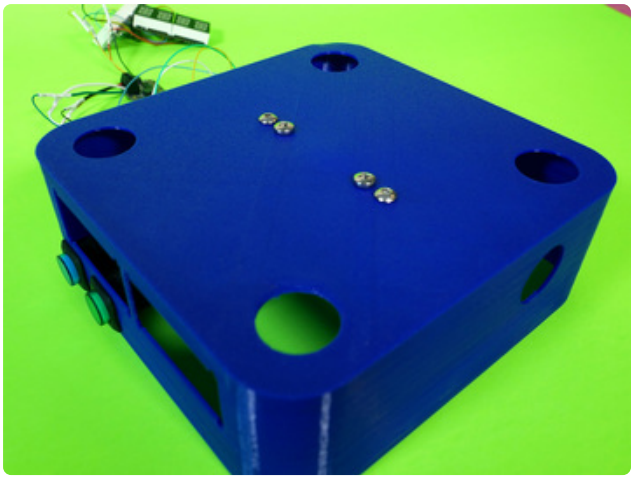
https://adafru.it/10bL

foodScaleSTLs_v1.zip

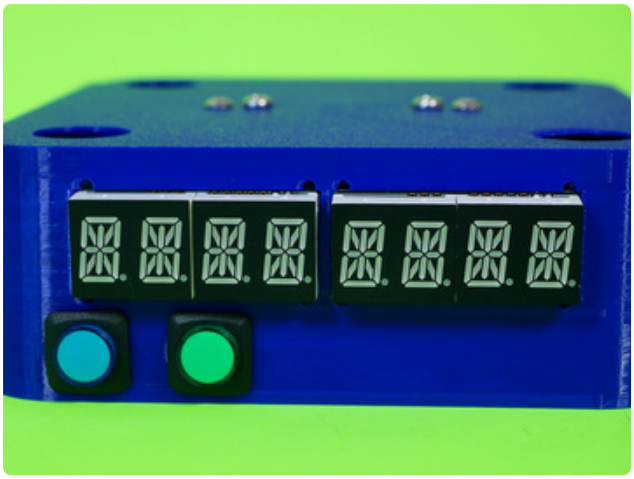
https://adafru.it/10c6



There are four stand-offs to use with the strain gauge and M4 screws so that the strain gauge can be mounted to both the top and bottom of the case.



The top and bottom of the case has mounting holes for the strain gauge and cutouts for your container's standoffs.



The buttons and displays mount in the front of the case.

CircuitPython

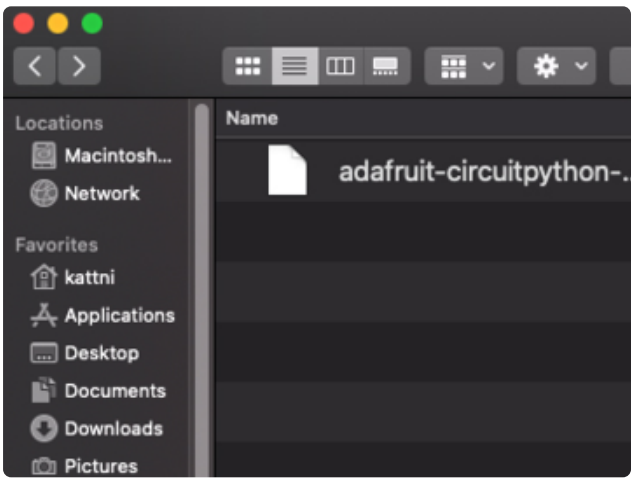
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

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

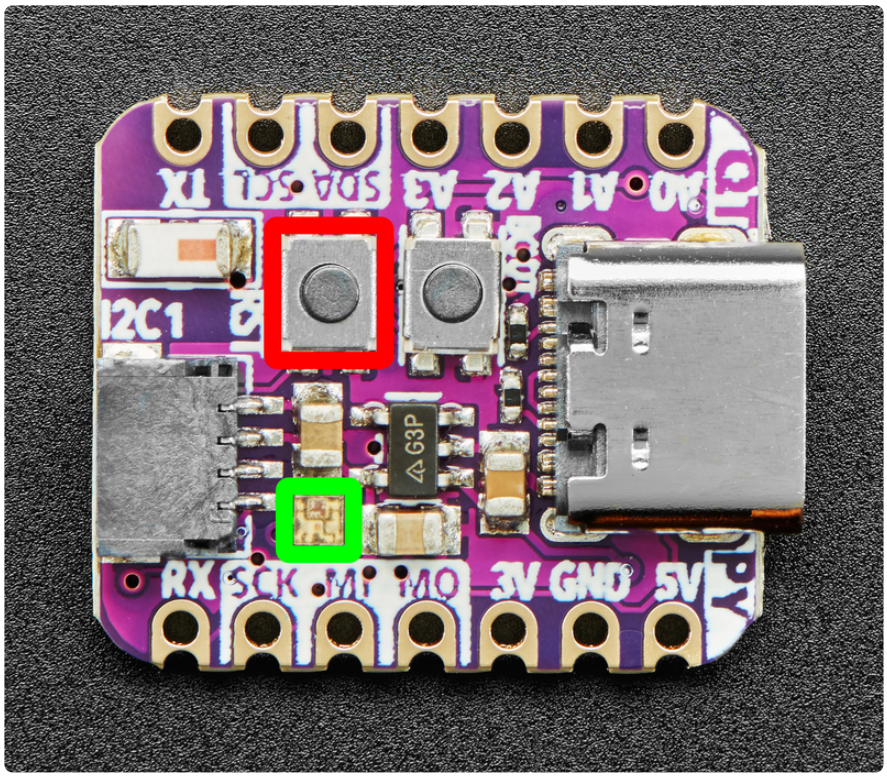
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/XCk



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

Save it wherever is convenient for you.



The board above has a chip antenna, not the u.FI connector, but the process is the same.

Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Click the **reset** button once (highlighted in red above), and then click it again when you see the **RGB status LED(s)** (highlighted in green above) turn purple (approximately half a second later). Sometimes it helps to think of it as a "slow double-click" of the reset button.

If you do not see the LED turning purple, you will need to reinstall the UF2 bootloader. See the **Factory Reset** page in this guide for details.

On some very old versions of the UF2 bootloader, the status LED turns red instead of purple.

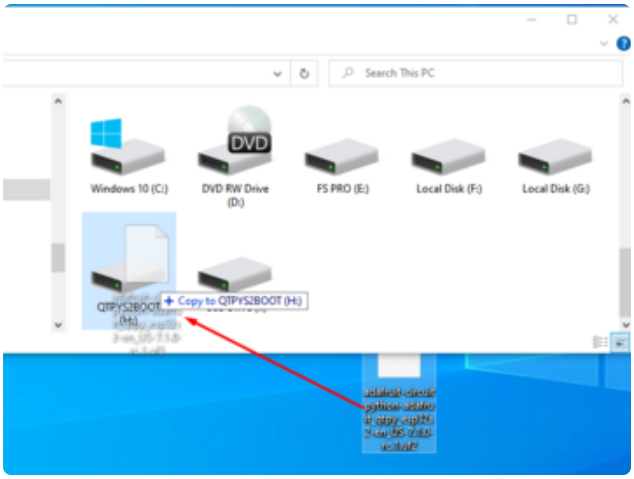
For this board, tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again. The second tap needs to happen while the LED is still purple.

Once successful, you will see the **RGB status LED(s)** turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

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

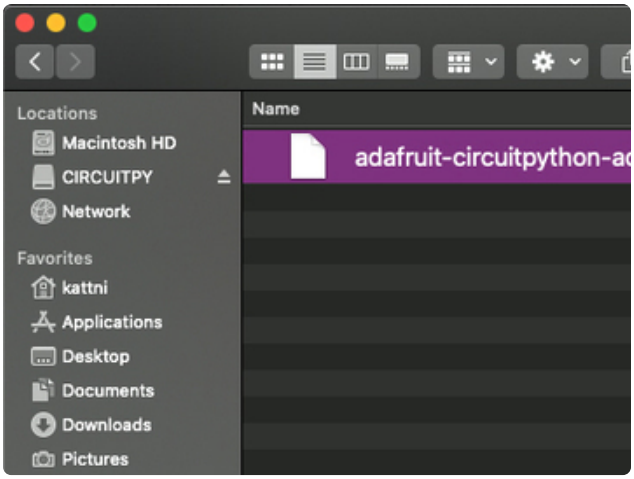
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**

If after several tries, and verifying your USB cable is data-ready, you still cannot get to the bootloader, it is possible that the bootloader is missing or damaged. Check out the Factory Reset page for details on resolving this issue.



You will see a new disk drive appear called **QTPYS2BOOT**.

Drag the **adafruit_circuitpython_etc.uf2** file to **QTPYS2BOOT**.



The **BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it!

Code the NAU7802 Food Scale

Once you've finished setting up your QT Py ESP32-S2 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.

```
# SPDX-FileCopyrightText: 2022 Liz Clark for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
from digitalio import DigitalInOut, Direction, Pull
from adafruit_ht16k33.segments import Seg14x4
from cedargrove_nau7802 import NAU7802
from calibration import calibration

#  I2C setup with STEMMA port
i2c = board.STEMMA_I2C()
#  alphanumeric segment displpay setup
#  using two displays together
display = Seg14x4(i2c, address=(0x70, 0x71))
#  start-up text
display.print("*HELLO* ")
#  button LEDs
```

```
blue = DigitalInOut(board.A1)
blue.direction = Direction.OUTPUT
green = DigitalInOut(board.A3)
green.direction = Direction.OUTPUT
#  buttons setup
blue_btn = DigitalInOut(board.A0)
blue_btn.direction = Direction.INPUT
blue_btn.pull = Pull.UP
green_btn = DigitalInOut(board.A2)
green_btn.direction = Direction.INPUT
green_btn.pull = Pull.UP
# nau7802 setup
nau7802 = NAU7802(board.STEMMA_I2C(), address=0x2A, active_channels=2)
nau7802.gain = 128
enabled = nau7802.enable(True)

#  zeroing function
def zero_channel():
    """Initiate internal calibration for current channel; return raw zero
    offset value. Use when scale is started, a new channel is green_btned, or to
    adjust for measurement drift. Remove weight and tare from load cell before
    executing."""
    blue.value = True
    print(
        "channel %1d calibrate.INTERNAL: %5s"
        % (nau7802.channel, nau7802.calibrate("INTERNAL"))
    )
    blue.value = False
    print(
        "channel %1d calibrate.OFFSET:   %5s"
        % (nau7802.channel, nau7802.calibrate("OFFSET"))
    )
    blue.value = True
    zero_offset = read_raw_value(100)  # Read 100 samples to establish zero offset
    print("...channel %1d zeroed" % nau7802.channel)
    blue.value = False
    return zero_offset
#  read raw value function
def read_raw_value(samples=100):
    """Read and average consecutive raw sample values. Return average raw value."""
    sample_sum = 0
    sample_count = samples
    while sample_count > 0:
        if nau7802.available:
            sample_sum = sample_sum + nau7802.read()
            sample_count -= 1
    return int(sample_sum / samples)
#  function for finding the average of an array
def find_average(num):
    count = 0
    for n in num:
        count = count + n
    average = count / len(num)
    return average
#  calibration function
def calculateCalibration(array):
    for _ in range(10):
        blue.value = True
        green.value = False
        nau7802.channel = 1
        #value = read_raw_value()
        print("channel %1.0f raw value: %7.0f" % (nau7802.channel,
abs(read_raw_value())))
        array.append(abs(read_raw_value()))
        blue.value = False
        green.value = True
        time.sleep(1)
    green.value = False
    avg = find_average(array)
```

```
    return avg
#  blink LED function
def blink(led, amount, count):
    for _ in range(count):
        led.value = True
        time.sleep(amount)
        led.value = False
        time.sleep(amount)

#  zeroing on startup
display.fill(0)
display.marquee("CLEAR SCALE CLEAR", 0.3, False)
time.sleep(2)
display.fill(0)
display.print("ZEROING")
time.sleep(3)
#  zeroing each channel
nau7802.channel = 1
zero_channel()  # Calibrate and zero channel
display.fill(0)
display.print("STARTING")

#  variables and states
clock = time.monotonic() #  time.monotonic() device
reset_clock = time.monotonic()
long_clock = time.monotonic()
mode = "run"
mode_names = ["SHOW OZ?", "   GRAMS?", "   ZERO?", "CALIBRTE", " OFFSET?"]
stage = 0
zero_stage = 0
weight_avg = 0
zero_avg = 0
show_oz = True
show_grams = False
zero_out = False
calibrate_mode = False
blue_btn_pressed = False
green_btn_pressed = False
run_mode = True
avg_read = []
values = []
val_offset = 0
avg_values = []

for w in range(5):
    nau7802.channel = 1
    value = read_raw_value()
    #  takes value reading and divides with by the offset value
    #  to get the weight in grams
    grams = value / calibration['offset_val']
    avg_read.append(grams)
    if len(avg_read) > 4:
        the_avg = find_average(avg_read)
        oz = the_avg / 28.35
        display.print("   %0.1f oz" % oz)
        avg_read.clear()
    time.sleep(1)

while True:
    #  button debouncing
    if blue_btn.value and blue_btn_pressed:
        blue_btn_pressed = False
    if green_btn.value and green_btn_pressed:
        green_btn_pressed = False
        green.value = False
    #  default run mode
    #  checks NAU7802 every 2 seconds
    if run_mode is True and (time.monotonic() - clock) > 2:
        nau7802.channel = 1
```

```
        value = read_raw_value()
        print(value)
        value = abs(value) - val_offset
        print(value)
        #value = abs(value)
        values.append(value)
        #  takes value reading and divides with by the offset value
        #  to get the weight in grams
        grams = value / calibration['offset_val']
        oz = grams / 28.35
        if show_oz is True:
            #  append reading
            avg_read.append(oz)
            label = "oz"
        if show_grams is True:
            avg_read.append(grams)
            label = "g"
        print(avg_read)
        if len(avg_read) > 10:
            the_avg = find_average(avg_read)
            display.print("   %0.1f %s" % (the_avg, label))
            avg_read.clear()
            val_offset += 10
        clock = time.monotonic()
    if (time.monotonic() - reset_clock) > 43200:
        run_mode = False
        show_oz = False
        show_grams = False
        zero_out = True
        reset_clock = time.monotonic()
    #  if you press the change mode button
    if (not green_btn.value and not green_btn_pressed) and run_mode:
        green.value = True
        #  disables run mode (stops weighing)
        run_mode = False
        show_oz = False
        show_grams = False
        #  mode is set to 0
        mode = 0
        #  display shows the mode option
        display.print(mode_names[mode])
        blue.value = True
        green_btn_pressed = True
    #  advances through the modes menu
    if (not green_btn.value and not green_btn_pressed) and mode != "run":
        green.value = True
        #  counts up to 4 and loops back to 0
        mode = (mode+1) % 5
        #  updates display
        display.print(mode_names[mode])
        green_btn_pressed = True
    #  if you select show_oz
    if (not blue_btn.value and not blue_btn_pressed) and mode == 0:
        #  show_oz is set as the state
        show_oz = True
        label = "oz"
        blue.value = False
        #  goes back to weighing mode
        mode = "run"
        blue_btn_pressed = True
        display.print("   %0.1f %s" % (the_avg, label))
        run_mode = True
    #  if you select show_grams
    if (not blue_btn.value and not blue_btn_pressed) and mode == 1:
        #  show_grams is set as the state
        show_grams = True
        label = "g"
        blue.value = False
        #  goes back to weighing mode
```

```
        mode = "run"
        blue_btn_pressed = True
        display.print("   %0.1f %s" % (the_avg, label))
        run_mode = True
    #  if you select zero_out
    if (not blue_btn.value and not blue_btn_pressed) and mode == 2:
        #  zero_out is set as the state
        #  can zero out the scale without full recalibration
        zero_out = True
        blue.value = False
        mode = "run"
        blue_btn_pressed = True
    #  if you select calibrate_mode
    if (not blue_btn.value and not blue_btn_pressed) and mode == 3:
        #  calibrate_mode is set as the state
        #  starts up the calibration process
        calibrate_mode = True
        blue.value = False
        mode = "run"
        blue_btn_pressed = True
    #  if you select the offset
    if (not blue_btn.value and not blue_btn_pressed) and mode == 4:
        #  displays the curren offset value stored in the code
        blue.value = False
        display.fill(0)
        display.print("%0.4f" % calibration['offset_val'])
        time.sleep(5)
        mode = "run"
        #  goes back to weighing mode
        show_oz = True
        label = "oz"
        display.print("   %0.1f %s" % (the_avg, label))
        run_mode = True
        blue_btn_pressed = True
    #  if the zero_out state is true
    if zero_out and zero_stage == 0:
        blue_btn_pressed = True
        #  clear the scale for zeroing
        display.fill(0)
        display.print("REMOVE ")
        zero_stage = 1
        blue.value = True
        green.value = True
    if (not blue_btn.value and not blue_btn_pressed) and zero_stage == 1:
        green.value = False
        #  updates display
        display.fill(0)
        display.print("ZEROING")
        blue.value = False
        #  runs zero_channel() function on both channels
        nau7802.channel = 1
        zero_channel()
        display.fill(0)
        display.print("ZEROED ")
        zero_out = False
        zero_stage = 0
        #  goes into weighing mode
        val_offset = 0
        run_mode = True
        show_oz = True
        label = "oz"
        display.print("   %0.1f %s" % (the_avg, label))
    #  the calibration process
    #  each step is counted in stage
    #  blue button is pressed to advance to the next stage
    if calibrate_mode is True and stage == 0:
        blue_btn_pressed = True
        #  clear the scale for zeroing
        display.fill(0)
```

```
        display.print("REMOVE ")
        stage = 1
        blue.value = True
    #  stage 2
    if (not blue_btn.value and not blue_btn_pressed) and stage == 1:
        blue_btn_pressed = True
        #  runs the zero out function
        display.fill(0)
        display.print("ZEROING")
        blue.value = False
        nau7802.channel = 1
        zero_channel()
        display.fill(0)
        display.print("ZEROED ")
        stage = 2
        blue.value = True
    #  stage 3
    if (not blue_btn.value and not blue_btn_pressed) and stage == 2:
        blue_btn_pressed = True
        blue.value = False
        display.print("STARTING")
        blink(blue, 0.5, 3)
        zero_readings = []
        display.print("AVG ZERO")
        #  runs the calculateCallibration function
        #  takes 10 raw readings, stores them into an array and gets an average
        zero_avg = calculateCalibration(zero_readings)
        stage = 3
        display.fill(0)
        display.print("DONE")
        blue.value = True
    #  stage 4
    if (not blue_btn.value and not blue_btn_pressed) and stage == 3:
        #  place the known weight item
        #  item's weight matches calibration['weight'] in grams
        blue_btn_pressed = True
        blue.value = False
        display.fill(0)
        display.print("PUT ITEM")
        stage = 4
        blue.value = True
    #  stage 5
    if (not blue_btn.value and not blue_btn_pressed) and stage == 4:
        blue_btn_pressed = True
        blue.value = False
        display.fill(0)
        display.print("WEIGHING")
        weight_readings = []
        #  weighs the item 10 times, stores the readings in an array & averages them
        weight_avg = calculateCalibration(weight_readings)
        #  calculates the new offset value
        calibration['offset_val'] = (weight_avg-zero_avg) / calibration['weight']
        display.marquee("%0.2f - CALIBRATED " % calibration['offset_val'], 0.3,
False)
        stage = 5
        display.fill(0)
        display.print("DONE")
        blue.value = True
    #  final stage
    if (not blue_btn.value and not blue_btn_pressed) and stage == 5:
        blue_btn_pressed = True
        zero_readings.clear()
        weight_readings.clear()
        calibrate_mode = False
        blue.value = False
        #  goes back into weighing mode
        show_oz = True
        label = "oz"
        display.print("   %0.1f %s" % (the_avg, label))
```

```
val_offset = 0
run_mode = True
#  resets stage
stage = 0
```

Upload the Code and Libraries to the QT Py ESP32-S2

After downloading the Project Bundle, plug your QT Py ESP32-S2 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 ESP32-S2's **CIRCUITPY** drive.

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

Your QT Py ESP32-S2 **CIRCUITPY** drive should look like this after copying the **lib** folder, **calibration.py** file and the **code.py** file.

Install the **cedargrove_nau7802** CircuitPython Library

Follow along with the steps outlined in this guide (https://adafru.it/10bN) to download the **cedargrove_nau7802** CircuitPython library and upload it to your QT Py ESP32-S2 **CIRCUITPY** drive **lib** folder. The library is a part of the CircuitPython Community Bundle (https://adafru.it/ENC).

calibration.py File

The **calibration.py** file holds two important values for the main **code.py** file: `offset_val` and `weight`. `weight` is the known weight in grams that you use to

calibrate the NAU7802. The `offset_val` is the calibration number used to divide against the raw value from the NAU7802 to find the actual weight on top of the scale.

Each strain gauge is slightly different, so you'll want to edit the **calibration.py** file with your known weight item and the `offset_val` you generate after running the calibration mode in the **code.py** file.

How the CircuitPython Code Works

First, the LED segment displays, buttons, button LEDs and NAU7802 are setup.

```
I2C setup with STEMMA port
i2c = board.STEMMA_I2C()
#  alphanumeric segment displpay setup
#  using two displays together
display = Seg14x4(i2c, address=(0x70, 0x71))
#  start-up text
display.print("*HELLO* ")
#  button LEDs
blue = DigitalInOut(board.A1)
blue.direction = Direction.OUTPUT
green = DigitalInOut(board.A3)
green.direction = Direction.OUTPUT
#  buttons setup
blue_btn = DigitalInOut(board.A0)
blue_btn.direction = Direction.INPUT
blue_btn.pull = Pull.UP
green_btn = DigitalInOut(board.A2)
green_btn.direction = Direction.INPUT
green_btn.pull = Pull.UP
# nau7802 setup
nau7802 = NAU7802(board.STEMMA_I2C(), address=0x2A, active_channels=2)
nau7802.gain = 128
enabled = nau7802.enable(True)
```

Functions

There are a few functions for the NAU7802. `zero_channel()` is from the **cedargrove_nau7802** library and resets the NAU7802 to be zeroed out. The blue button's LED blinks as the zeroing process takes place.

```
#  zeroing function
def zero_channel():
    """Initiate internal calibration for current channel; return raw zero
    offset value. Use when scale is started, a new channel is green_btned, or to
    adjust for measurement drift. Remove weight and tare from load cell before
    executing."""
    blue.value = True
    print(
        "channel %1d calibrate.INTERNAL: %5s"
        % (nau7802.channel, nau7802.calibrate("INTERNAL"))
    )
    blue.value = False
    print(
        "channel %1d calibrate.OFFSET:   %5s"
```

```
        % (nau7802.channel, nau7802.calibrate("OFFSET"))
    )
    blue.value = True
    zero_offset = read_raw_value(100)  # Read 100 samples to establish zero offset
    print("...channel %1d zeroed" % nau7802.channel)
    blue.value = False
    return zero_offset
```

`read_raw_value()` reads the raw value from the NAU7802 and is also from the **cedargrove_nau7802** library.

```
#  read raw value function
def read_raw_value(samples=100):
    """Read and average consecutive raw sample values. Return average raw value."""
    sample_sum = 0
    sample_count = samples
    while sample_count &gt; 0:
        if nau7802.available:
            sample_sum = sample_sum + nau7802.read()
            sample_count -= 1
    return int(sample_sum / samples)
```

`find_average(num)` is a quick function for finding the average of an array of values.

```
#  function for finding the average of an array
def find_average(num):
    count = 0
    for n in num:
        count = count + n
    average = count / len(num)
    return average
```

`calculateCalibration(array)` is the function used for calibrating the NAU7802 to find the `offset_val`. It runs `read_raw_value()` ten times and appends those values to an array. That array is averaged and the average is returned.

```
def calculateCalibration(array):
    for _ in range(10):
        blue.value = True
        green.value = False
        nau7802.channel = 1
        #value = read_raw_value()
        print("channel %1.0f raw value: %7.0f" % (nau7802.channel,
abs(read_raw_value())))
        array.append(abs(read_raw_value()))
        blue.value = False
        green.value = True
        time.sleep(1)
    green.value = False
    avg = find_average(array)
    return avg
```

`blink(led, amount, count)` is a quick function for blinking an LED. You pass the LED's pin for `led`, length of time for each blink for `amount` and number of blinks for `count`.

```
#  blink LED function
def blink(led, amount, count):
    for _ in range(count):
        led.value = True
        time.sleep(amount)
        led.value = False
        time.sleep(amount)
```

Startup Reading

On startup, the NAU7802 is zeroed and then an initial reading is taken and displayed in ounces. The displays and button LEDs let you know what is happening in the code.

```
#  zeroing on startup
display.fill(0)
display.marquee("CLEAR SCALE CLEAR", 0.3, False)
time.sleep(2)
display.fill(0)
display.print("ZEROING")
time.sleep(3)
#  zeroing each channel
nau7802.channel = 1
zero_channel()  # Calibrate and zero channel
display.fill(0)
display.print("STARTING")

for w in range(5):
    nau7802.channel = 1
    value = read_raw_value()
    #  takes value reading and divides with by the offset value
    #  to get the weight in grams
    grams = value / calibration['offset_val']
    avg_read.append(grams)
    if len(avg_read) > 4:
        the_avg = find_average(avg_read)
        oz = the_avg / 28.35
        display.print("   %0.1f oz" % oz)
        avg_read.clear()
        averaging = 0
    time.sleep(1)
```

The Loop

The loop has multiple modes that can be selected with states. The default mode is `run_mode`. In `run_mode`, the NAU7802 takes a reading every two seconds. This value is converted to grams and ounces and is appended to the `avg_read` array.

Once the length of `avg_read` is greater than `10`, `avg_read` is averaged. The `return` is shown on the displays and then the process begins again. After this process runs `100` times, the scale prompts you to zero it out to retain accuracy.

The scale can show the weight in either ounces or grams, depending on whether `show_oz` or `show_grams` is `True`.

```
#  default run mode
    #  checks NAU7802 every 2 seconds
    if run_mode is True and (time.monotonic() - clock) &gt; 2:
        nau7802.channel = 1
        value = read_raw_value()
        value = abs(value) - val_offset
        #  takes value reading and divides with by the offset value
        #  to get the weight in grams
        grams = value / calibration['offset_val']
        oz = grams / 28.35
        if show_oz is True:
            #  append reading
            avg_read.append(oz)
            label = "oz"
        if show_grams is True:
            avg_read.append(grams)
            label = "g"
        if len(avg_read) &gt; 10:
            the_avg = find_average(avg_read)
            display.print("   %0.1f %s" % (the_avg, label))
            avg_read.clear()
            val_offset += 12
            count += 1
        if count &gt; 100:
            count = 0
            run_mode = False
            show_oz = False
            show_grams = False
            zero_out = True
        clock = time.monotonic()
```

Mode Select

If you press the green button, you get into mode select mode. This stops `run_mode` and displays the mode options on the displays. You can press the green button to cycle through the options and press the blue button to select a mode.

```
#  if you press the change mode button
    if (not green_btn.value and not green_btn_pressed) and run_mode:
        green.value = True
        #  disables run mode (stops weighing)
        run_mode = False
        show_oz = False
        show_grams = False
        #  mode is set to 0
        mode = 0
        #  display shows the mode option
        display.print(mode_names[mode])
        blue.value = True
        green_btn_pressed = True
```

Zeroing Mode

`show_oz` and `show_grams` change whether you're showing the weight in ounces or grams and goes back into `run_mode` once they're selected.

`zero_out` mode allows you to re-zero the NAU7802 at anytime. You'll be prompted to remove the weight from the scale, press the blue button and then `zero_channel()` runs.

```
#  if you select zero_out
    if (not blue_btn.value and not blue_btn_pressed) and mode == 2:
        #  zero_out is set as the state
        #  can zero out the scale without full recalibration
        zero_out = True
        blue.value = False
        mode = "run"
        blue_btn_pressed = True

#  if the zero_out state is true
    if zero_out and zero_stage == 0:
        blue_btn_pressed = True
        #  clear the scale for zeroing
        display.fill(0)
        display.print("REMOVE ")
        zero_stage = 1
        blue.value = True
        green.value = True
    if (not blue_btn.value and not blue_btn_pressed) and zero_stage == 1:
        green.value = False
        #  updates display
        display.fill(0)
        display.print("ZEROING")
        blue.value = False
        #  runs zero_channel() function on both channels
        nau7802.channel = 1
        zero_channel()
        display.fill(0)
        display.print("ZEROED ")
        zero_out = False
        stage = 0
        #  goes into weighing mode
        val_offset = 0
        run_mode = True
        show_oz = True
        label = "oz"
        display.print("   %0.1f %s" % (the_avg, label))
```

Show Offset Mode

By selecting show offset, the current `offset_val` is displayed for `5` seconds. This is helpful for updating your **calibration.py** file and other troubleshooting.

```
#  if you select the offset
    if (not blue_btn.value and not blue_btn_pressed) and mode == 4:
        #  displays the curren offset value stored in the code
        blue.value = False
        display.fill(0)
        display.print("%0.4f" % calibration['offset_val'])
        time.sleep(5)
        mode = "run"
        #  goes back to weighing mode
        show_oz = True
        label = "oz"
        display.print("   %0.1f %s" % (the_avg, label))
```

```
        run_mode = True
        blue_btn_pressed = True
```

Calibration Mode

`calibration_mode` lets you recalibrate the NAU7802 by running through the `zero_channel()` function, `calculateCalibration()` function with the scale zeroed, `calculateCalibration()` function with the defined weight object on the scale and then finding the new `offset_val`.

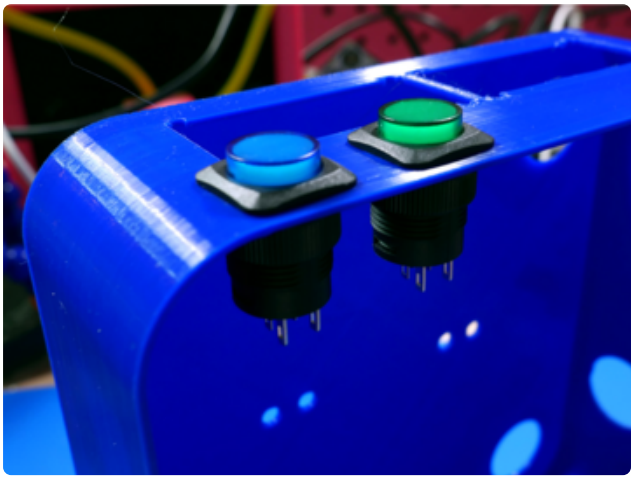
The steps of the process are advanced by button presses that trigger and increase the value of `stage`. This ensures that the scale is prepped for each stage of the calibration.

`calculateCalibration()` returns an average of the NAU7802's readings. The average of the scale with nothing on it (`zero_avg`) and the average of the scale with the defined weight object (`weight_avg`) are subtracted from each other and then divided by `weight`. The result of that sets the new `offset_val`.

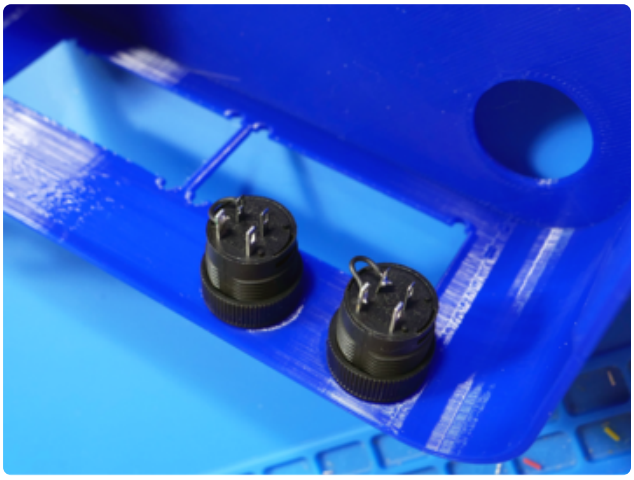
```
#  stage 5
    if (not blue_btn.value and not blue_btn_pressed) and stage == 4:
        blue_btn_pressed = True
        blue.value = False
        display.fill(0)
        display.print("WEIGHING")
        weight_readings = []
        #  weighs the item 10 times, stores the readings in an array &amp;amp; averages
them
        weight_avg = calculateCalibration(weight_readings)
        #  calculates the new offset value
        calibration['offset_val'] = (weight_avg-zero_avg) / calibration['weight']
        display.marquee("%0.2f - CALIBRATED " % calibration['offset_val'], 0.3,
False)
        stage = 5
        display.fill(0)
        display.print("DONE")
        blue.value = True
```

Wiring

Button Wiring

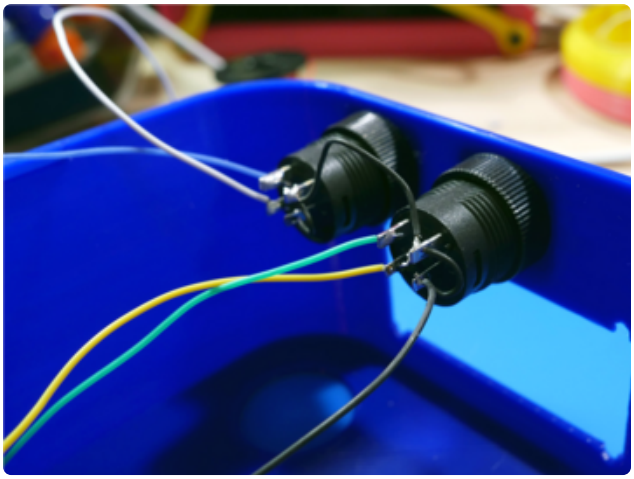


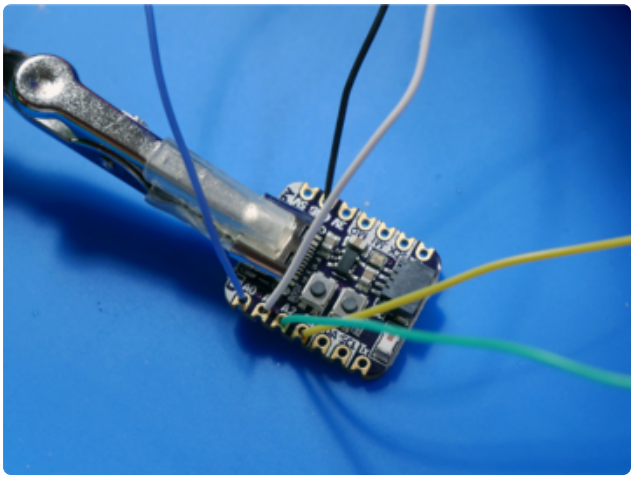
Mount the buttons into the front of the case.

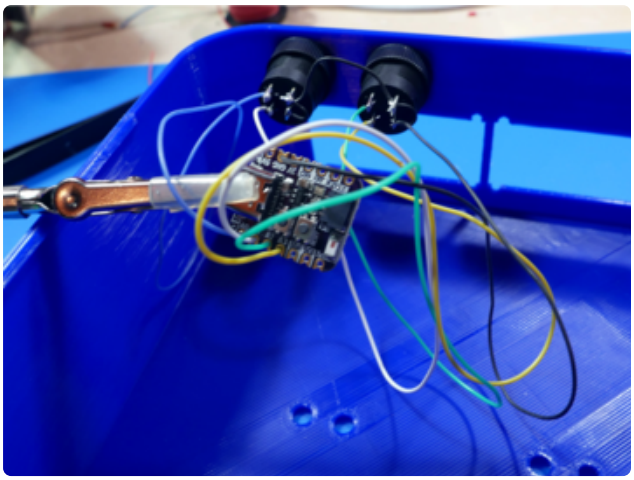


Solder each of the button LED GNDs and GND pins together with short pieces of wire. The LED GND pins are marked with a minus sign (-).

Solder the buttons' GND connections together.

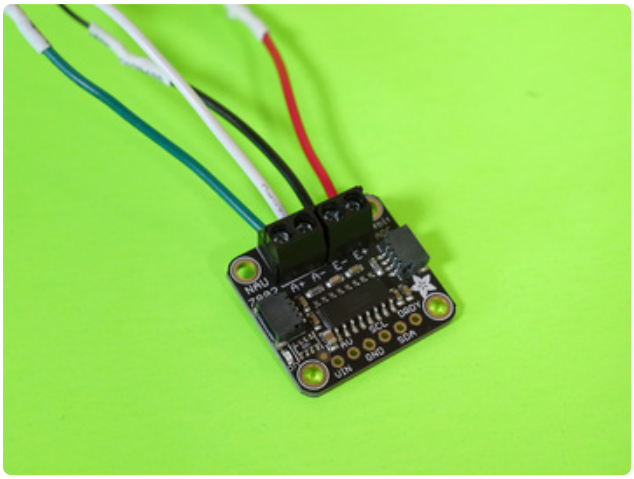






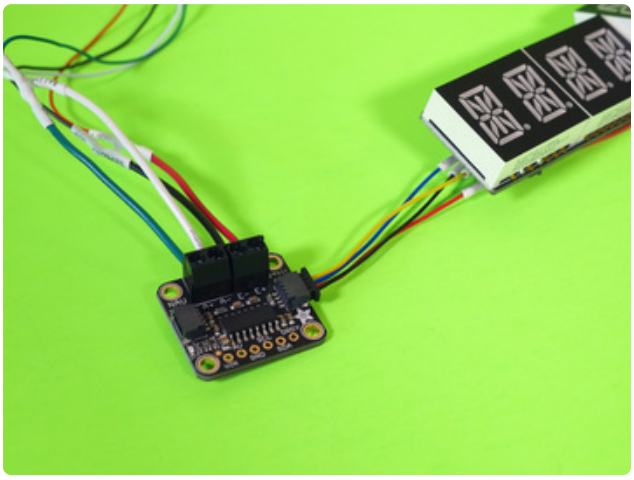
Connect the **blue button's LED** to the QT Py's **pin A0** (blue wire).
Connect the **blue button's input** to the QT Py's **pin A1** (white wire).
Connect the **green button's LED** to the QT Py's **pin A2** (green wire).
Connect the **green button's input** to the QT Py's **pin A3** (yellow wire).
Connect the **buttons' GND signal** to the QT Py's **GND pin** (black wire).

STEMMA Boards

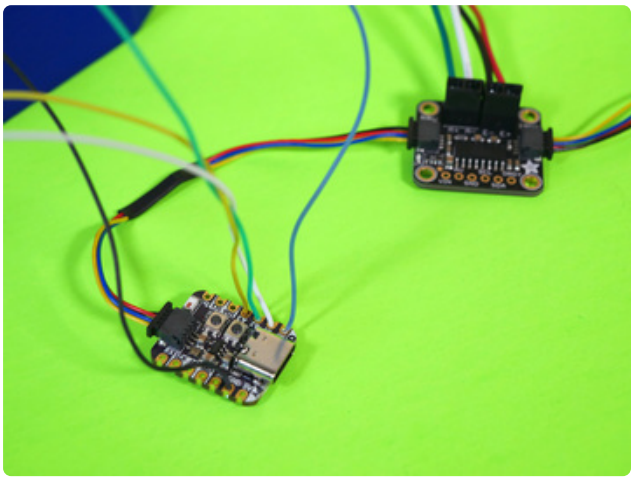


Insert the strain gauge's wires into the NAU7802's four terminal block connections:

Green wire to **A+**
White wire to **A-**
Black wire to **E-**
Red wire to **E+**

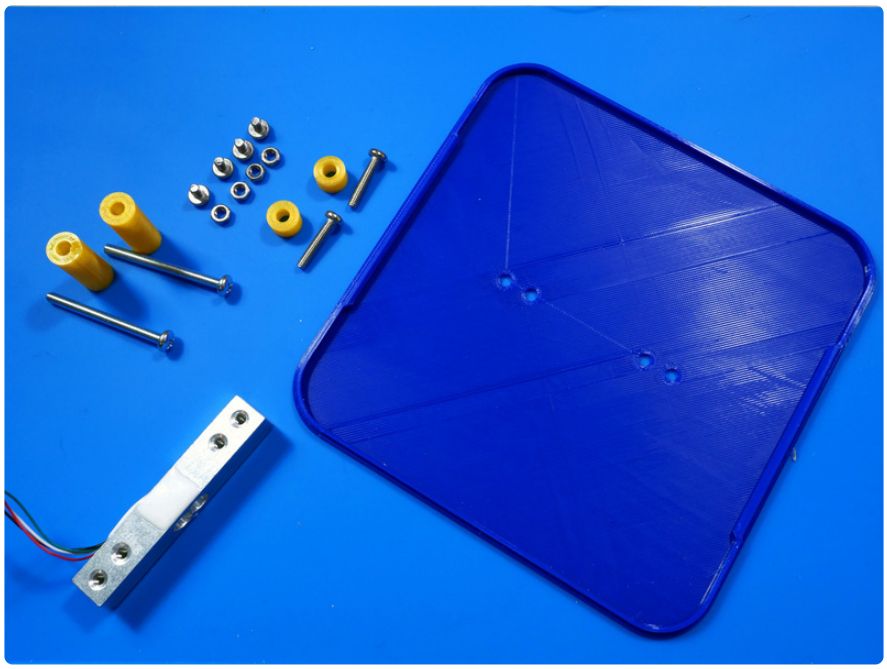


Connect the two alphanumeric display boards together with a STEMMA QT cable. Plug the displays into the NAU7802 with a STEMMA QT cable.



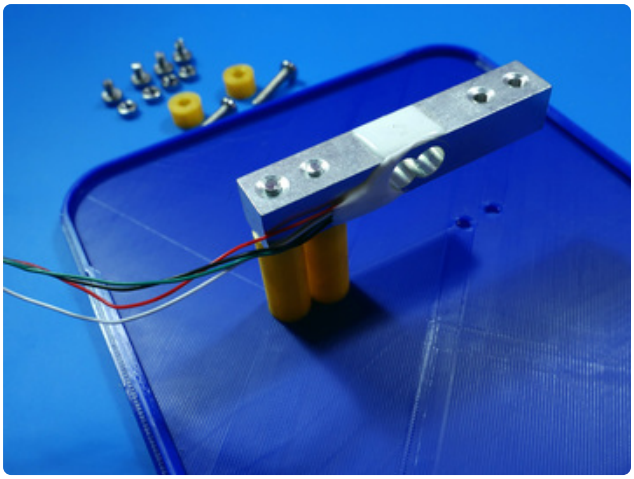
Plug the NAU7802 into the QT Py with a STEMMA QT cable. This completes all of the connections between the components.

Assembly

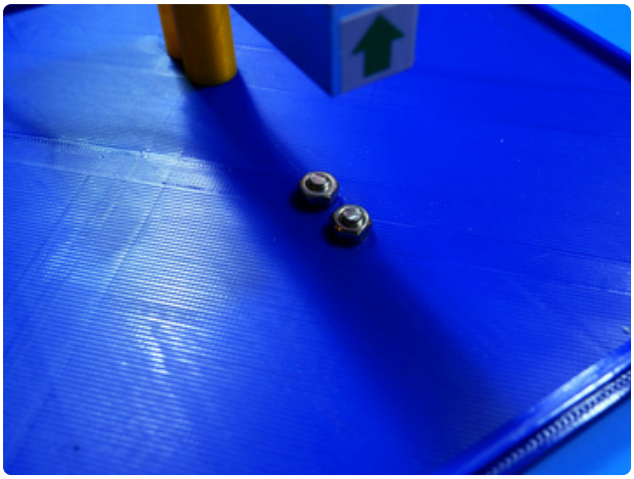


Mount the Strain Gauge

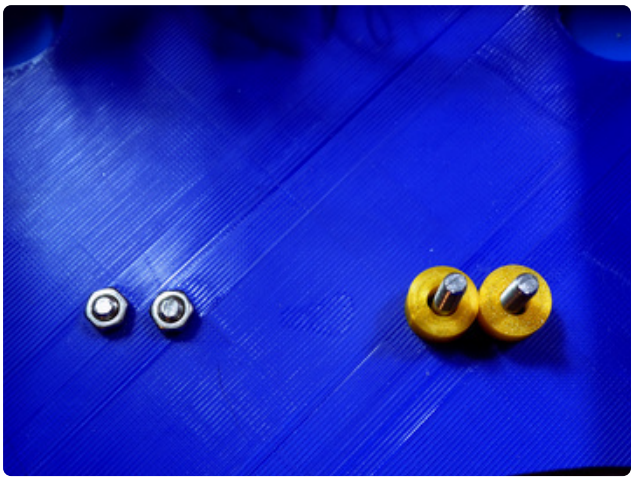
Insert two 45mm M4 screws with the long 3D printed stand-offs into the mounting holes on the left of the bottom lid.



Screw the 45mm screws into the bottom of the strain gauge.

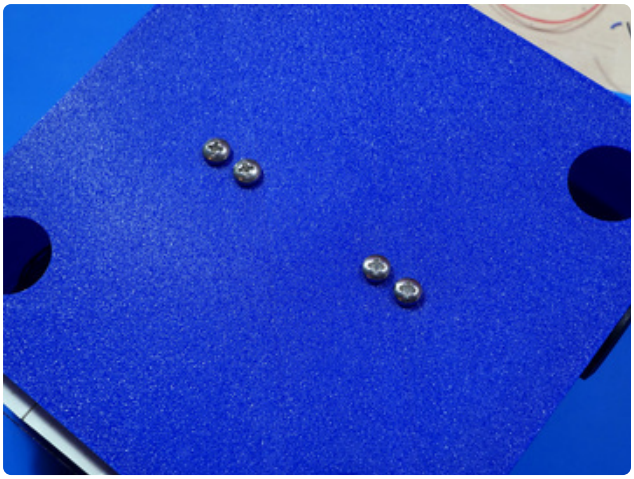


Attach two 4mm M4 screws to the empty mounting holes with M4 nuts. This will help with balancing the case on a surface.

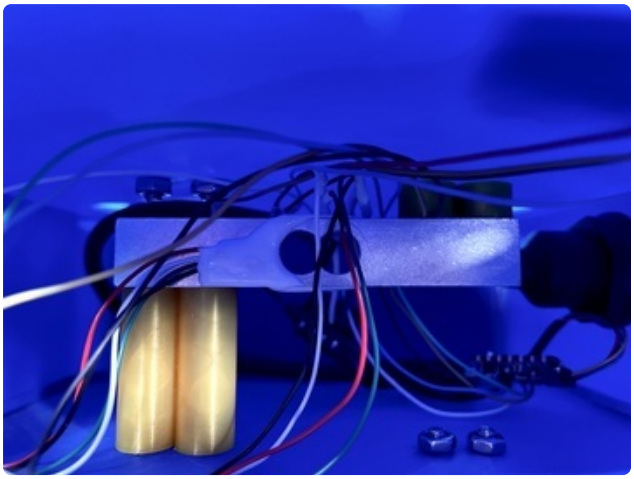


Attach two 4mm M4 screws with M4 nuts to the case lid's mounting holes on the left.

Insert two 20mm M4 screws with the short 3D printed stand-offs into the case lid's mounting holes on the right.

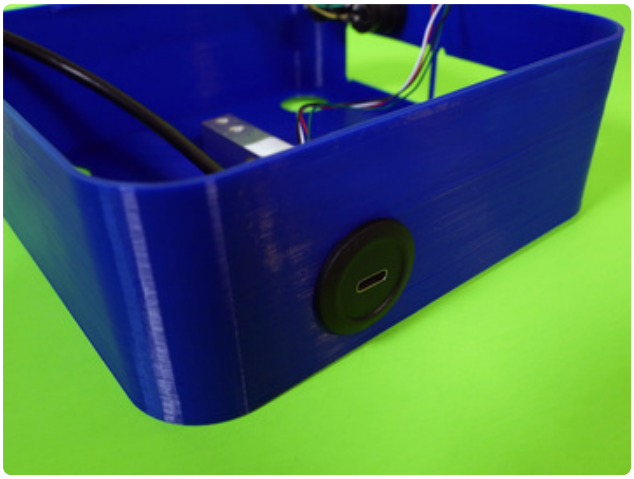


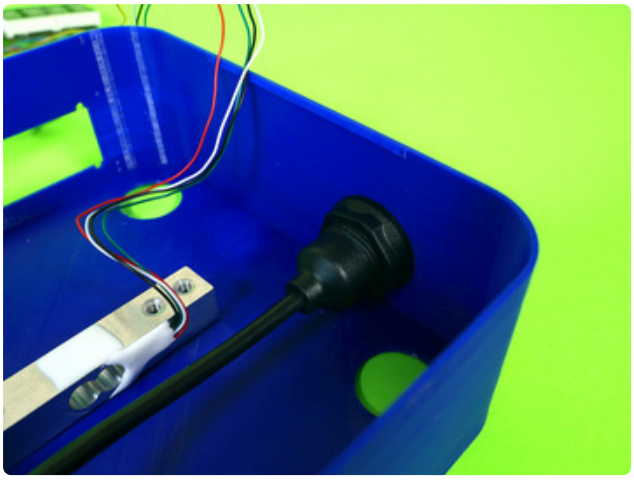
Screw the 20mm M4 screws with the stand-offs into the strain gauge.



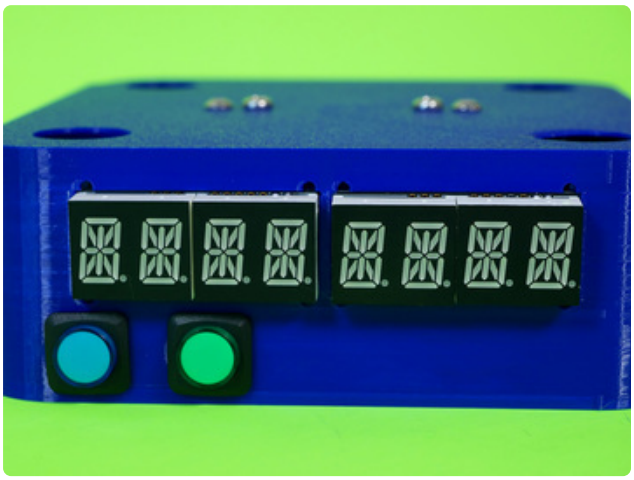
The right side of the strain gauge will be attached to the top of the case and the left side of the strain gauge will be attached to the bottom of the case. This allows the NAU7802 to measure torque as described in this article (https://adafru.it/10c7).

Mount the Electronics





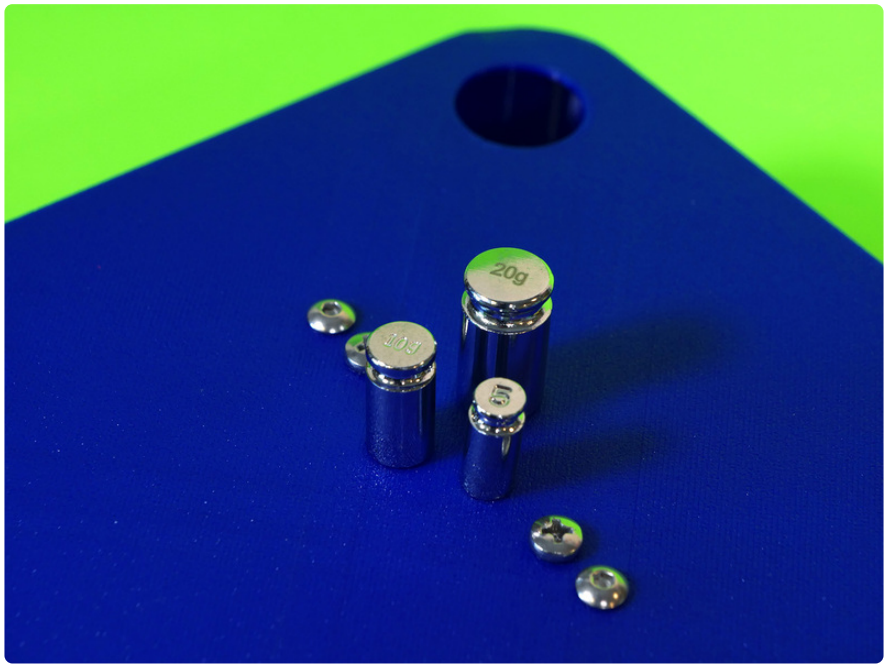
Mount the USB-C panel mount extension into the side of the case. Plug the extension into the QT Py's USB port.



Mount the two alphanumeric displays into the front of the case.

Close up the case with the bottom lid and you're ready to get weighing!

Calibration

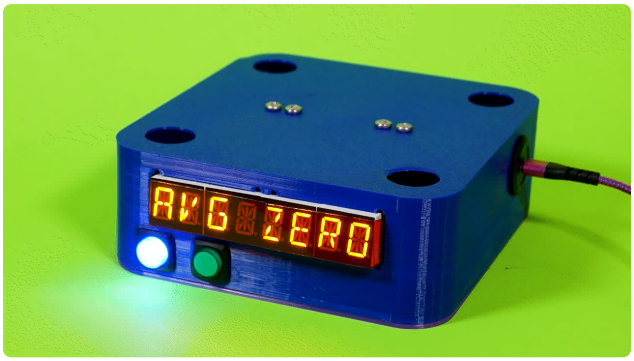


To get as accurate a measure as possible with the NAU7802 and a strain gauge, you'll need to go through a calibration process. To do this, you weigh something whose weight you are sure of, such as conveniently named calibration weights, and figure out the offset value from the sensor.

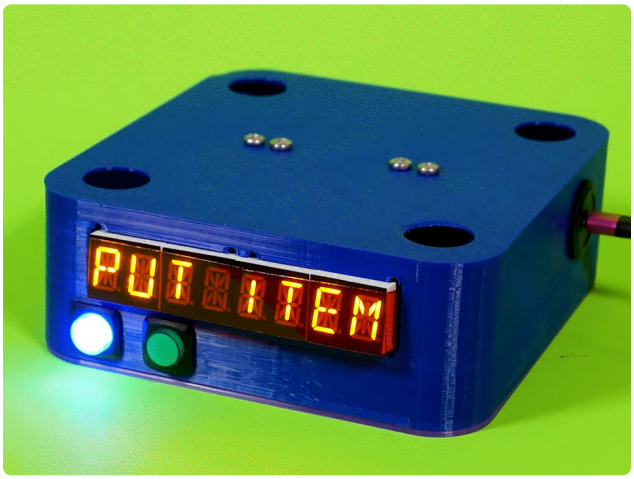
Essentially, you want to see what the raw value is from the sensor when the known weight is on top of it. Then, you know that that raw value is equal to a certain weight and can figure out the weight of other objects going forward.

To enter calibration mode, select calibration from the mode selection list. To start, you'll be prompted to clear the scale. Press the blue button when its cleared.

Then, the code will run the `zero_channel()` function to zero out the sensor. Press the blue button when it lights up.



Next, the code will get an average raw value from the NAU7802 with nothing on the scale.

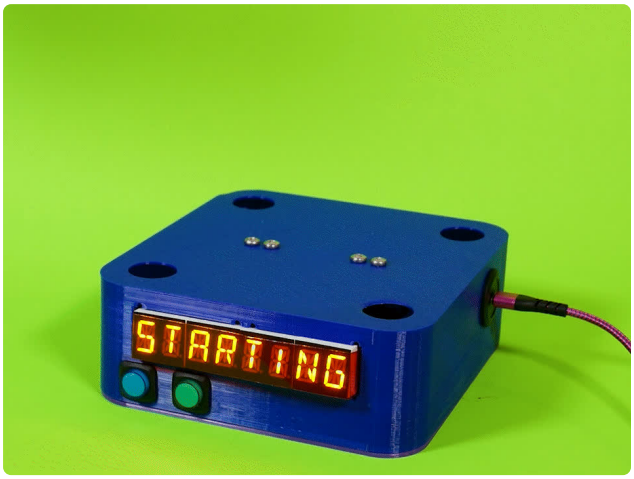


You'll be prompted to put your calibration weight item on the scale. When it's on the scale, press the blue button.

The code measures the average raw value with the calibration weight. Then, it calculates the new offset value based on the average zero measure and average calibration weight measure.

When it's done, the new offset value will scroll across the display. Then, you can press the blue button and get back to weighing things.

Usage



On boot up, the scale will zero itself. When it says "STARTING" on the display, you can place the food container filled with your pet's favorite munchies.

The display will show the weight of the food in ounces by default.

You can press the green button to cycle through the different modes. You can display the weight in grams, zero out the scale or run a full calibration.

You can also view your current offset value. This comes in handy for updating your **calibration.py** file after recalibrating.

Going further, you could try mounting the strain gauge to measure other object's weights, like laundry detergent or solder paste. You could also log the data over time to make this an IoT project.