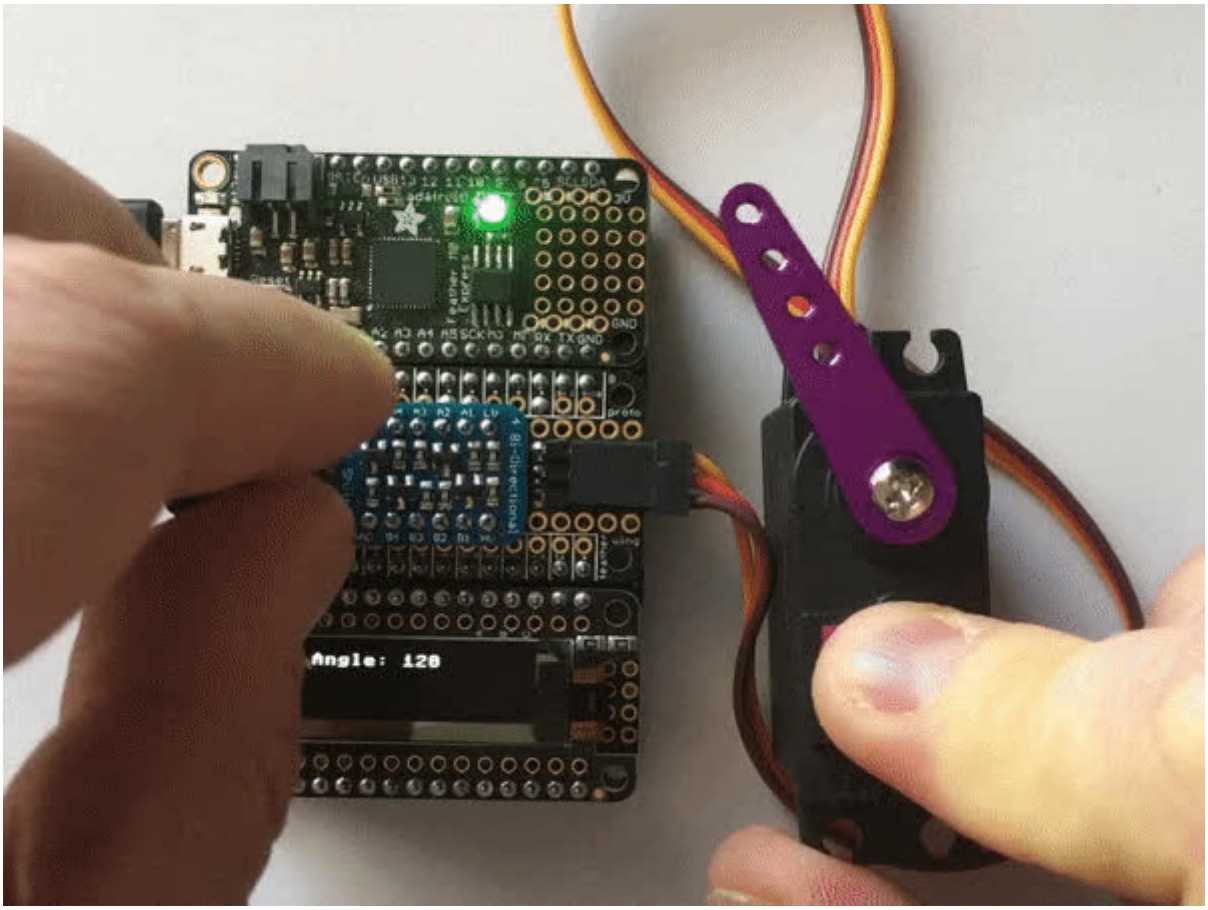
CircuitPython Servo Tester

Created by Dave Astels

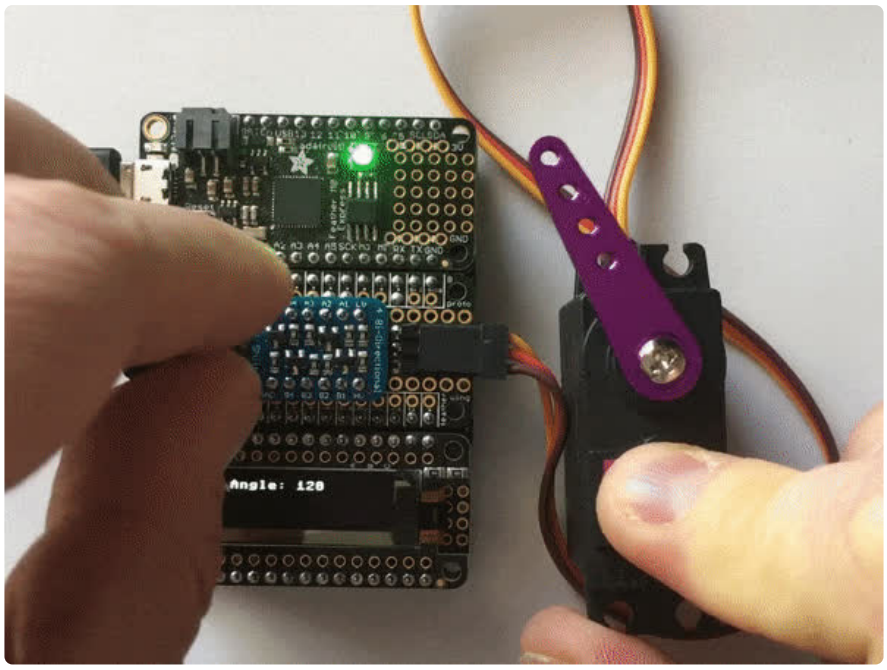


https://learn.adafruit.com/circuitpython-servo-tester

Last updated on 2024-06-03 02:33:19 PM EDT

Table of Contents

Overview

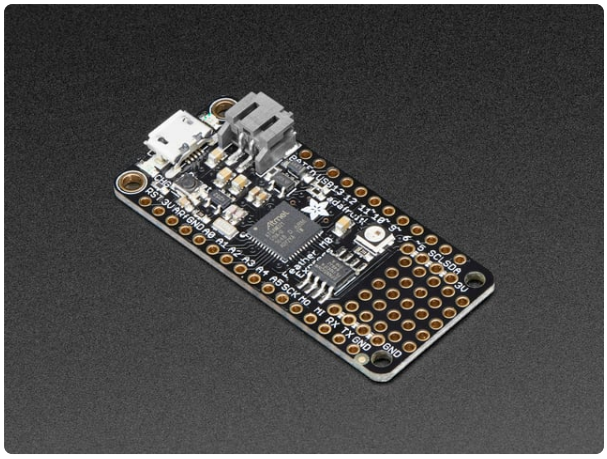


Servos are great fun, and incredibly useful when you want to make something move.

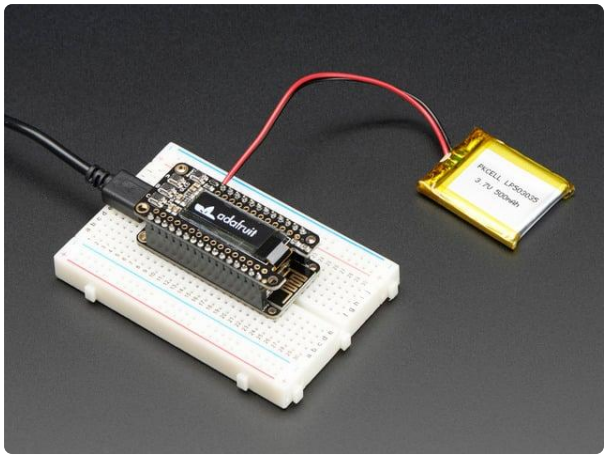
They do have some issues at times, though. Sometimes you need to set them to a specific angle for assembly. Turning the output shaft manually can be problematic since they're not made for that and on cheaper servos you can strip gears or break teeth pretty easily. If the servo already has broken teeth or such, it's motion can be erratic and it's best to know that before you put things together. Finally servos can vary a bit in what their pulse width bounds are. If you need a full range of motion from them you need to figure out and account for those bounds (Adafruit servo libraries allow you to set them when you construct a servo object).

All in all, it would be nice to have a way to fiddle around with a servo before building it into a project.

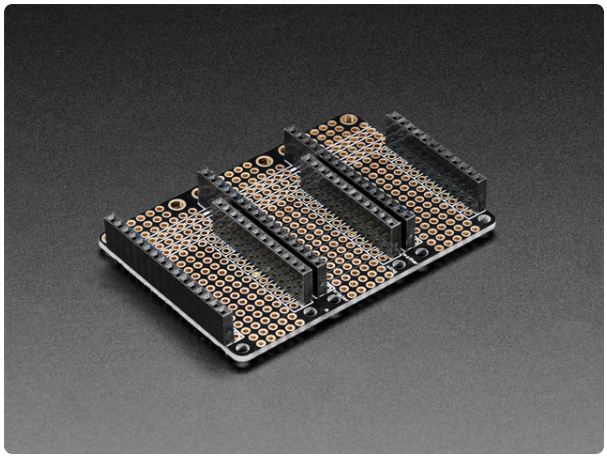
This project does just that. Hook up a servo and test it's range of motion (0-180 degrees), test the smoothness of its motion by having it sweep back and forth between 0 and 180 at different speeds, and try different boundary pulse widths to find the servo's limits.



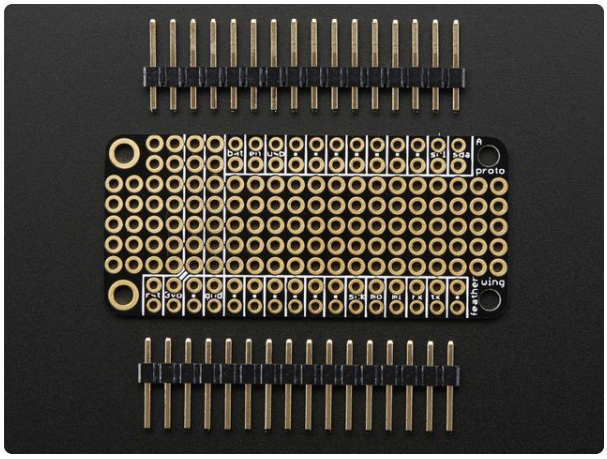
[Adafruit Feather M0 Express]
At the Feather M0's heart is an ATSAMD21G18 ARM Cortex M0+ processor, clocked at 48 MHz and at 3.3V logic, the same one used in the new
https://www.adafruit.com/product/3403



[Adafruit FeatherWing OLED - 128x32 OLED Add-on For Feather]
A Feather board without ambition is a Feather board without FeatherWings! This is the FeatherWing OLED: it adds a 128x32 monochrome OLED plus 3 user buttons to...
https://www.adafruit.com/product/2900



[FeatherWing Tripler Mini Kit - Prototyping Add-on For Feathers]
This is the FeatherWing Tripler - a prototyping add-on and more for all Feather boards. This is similar to our
https://www.adafruit.com/product/3417

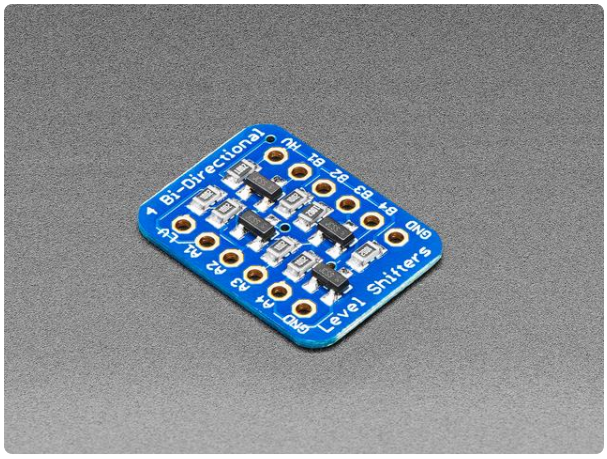


[FeatherWing Proto - Prototyping Add-on For All Feather Boards]
A Feather board without ambition is a Feather board without FeatherWings!This is the FeatherWing Proto - a prototyping add-on for all Feather boards. Using our...
https://www.adafruit.com/product/2884

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



4-channel I2C-safe Bi-directional Logic Level Converter

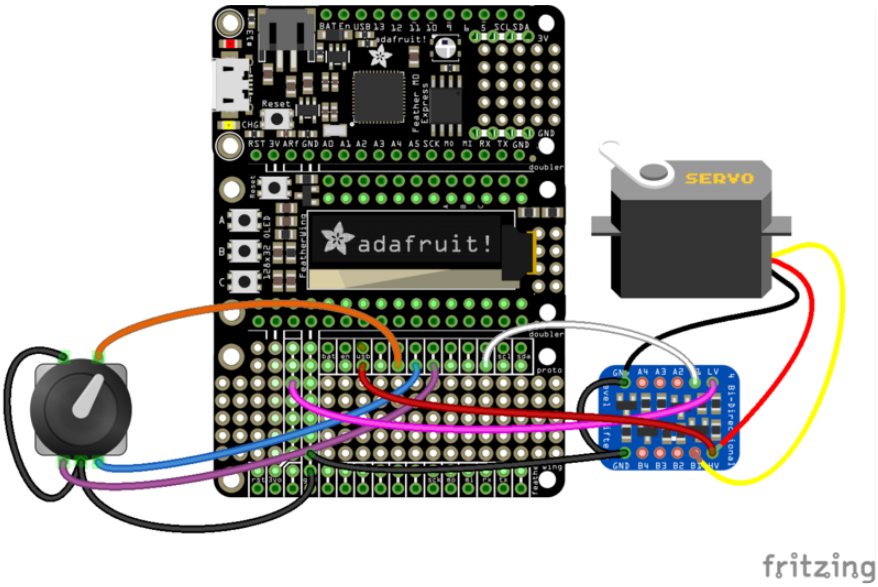
Because the Arduino (and Basic Stamp) are 5V devices, and most modern sensors, displays, flashcards, and modes are 3.3V-only, many makers find that they need to perform level...

https://www.adafruit.com/product/757

Other Tools and Supplies

- Soldering iron and solder
- hookup wire, the 30AWG Silicone Covered Stranded-Core Wire (http://adafru.it/3164) works great
- 3 pins of male header, either straight (http://adafru.it/2671) or right-angle (http://adafru.it/1540).

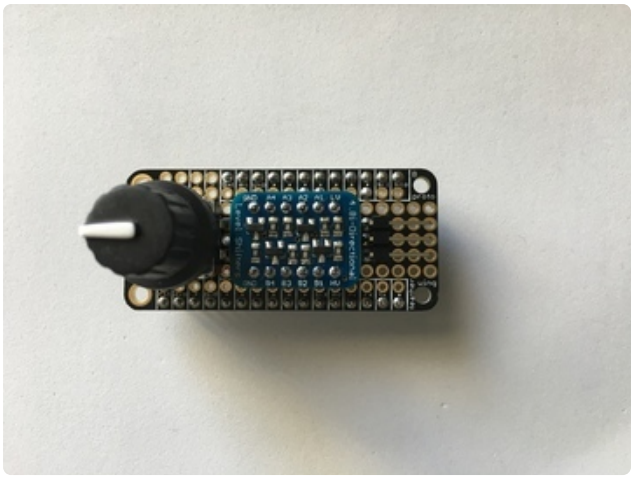
Circuitry



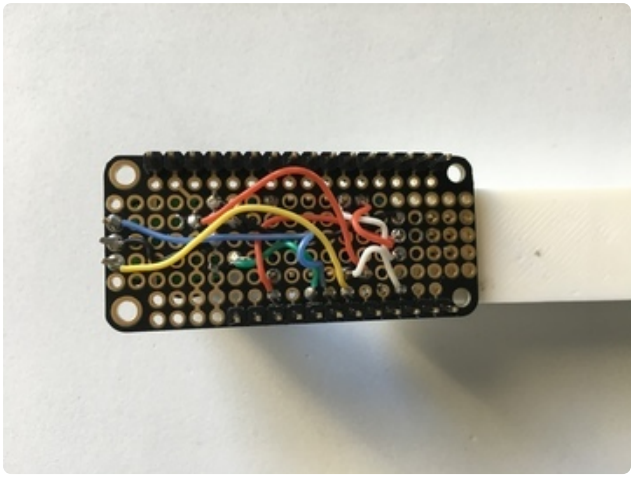
The circuit diagram above uses a Feather M0 Express and OLED breakout. Much of the wiring is taken care of by using a Feather and wings as we do here. All we need to build is a wing with the rotary encoder, level shifter, and servo header.

There's a level shifter to drive the servo with a 5v signal. This avoids any potential issue with a 3.3v signal not being a high enough voltage signal.

Using Feathers, wings, and a Feather doubler/tripler wing is a nice way to prototype a design even if you want make it more compact and put it in a case for a final build.



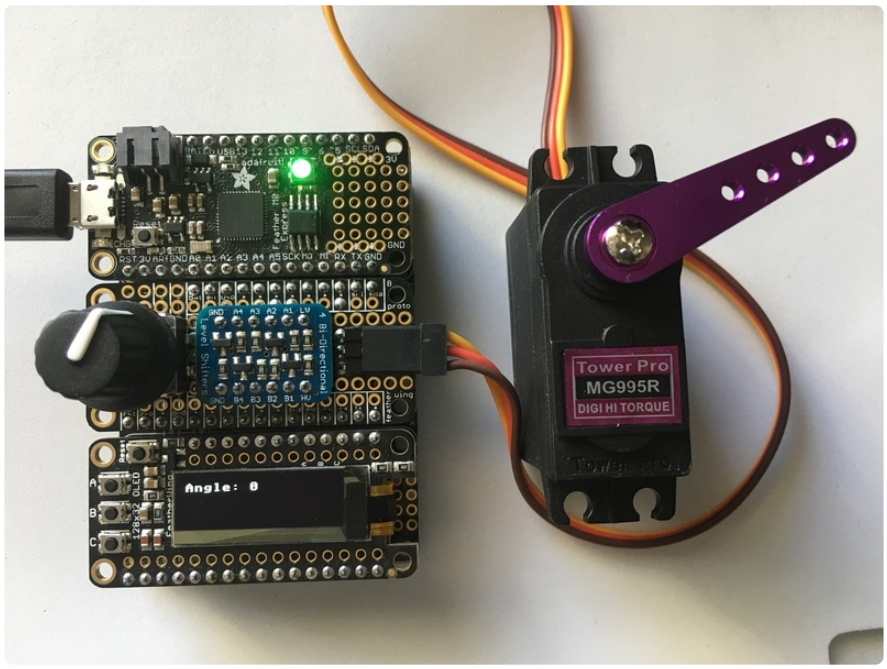
The rotary encoder fits nicely at the "usb" end of the proto wing, straddling the 3.3 and ground bus lines. The servo header goes at the other end, with the level shifter fitting neatly between them. If you use right angle header as shown here, there's just enough room to fit everything without the header pins extending beyond the edge of the wing.



Here, you can see the wiring. Colored wire is used:

red - both 3.3 to the level shifter and 5v lines to the shifter and servo
black - ground
yellow and blue - the two rotary inputs
green - the encoder switch input
white - the PWM signal for the servo (on both sides ot the level shifter)

Once the encoder/servo wing is built, snap it all onto the triple, connect a servo and power via USB.



Code

We'll be using CircuitPython for this project. Are you new to using CircuitPython? No worries, there is a full getting started guide here (https://adafru.it/cpy-welcome).

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and its installation in this tutorial (https://adafru.it/ANO).

Be sure to load CircuitPython on your board. See this tutorial (https://adafru.it/Amd) for the process.

This project requires CircuitPython 3.x or 4.0 alpha 1. CircuitPython 4.0 alpha 2 does not work due to ongoing evolution of the display code.

Like the circuit, the code is also straight-forward.

It works as a simple linear, looping state machine that moves to the next state each time the encoder button is pressed:

- Mode 0 - adjust the servo angle
- Mode 1 - sweep the servo repeated between 0 and 180 and back, allowing the time to sweep to be adjusted
- Mode 2 - adjust the lower bound of the pulse width
- Mode 3 - adjust the upper bound of the pulse width

To start it initializes hardware and sets up some variables, including a debouncer for the encoder switch. There's nothing special about this and it can be seen in the full listing below.

The main loop is made up of 6 sections.

Housekeeping

We start by grabbing the current time. This is the elapsed time actually, as we just need to measure differences in time, not absolute time of day. We also update the debouncer now.

```
    now = time.monotonic()
    button.update()
```

Sweep

Next, if the current mode is the sweep mode, the servo angle will need to be updated if the time between steps has passed. If the angle reaches either extreme, the direction of rotation is reversed.

```
    if mode == 1:
        if now &gt;= (last_movement_at + sweep_time / 36):
            last_movement_at = now
            angle += delta
            if (angle &gt; 180) or (angle &lt; 0):
                delta *= -1
                angle += delta
```

Mode

If the button was just pressed, we change the mode. If we are switching into mode 0 the angle is set to 0. If we are going into mode 1 we set the time it takes to sweep and next time to step the servo. If we are going into mode 2 or 3, the angle is set to the appropriate extreme.

```
    if button.fell:
        servo.angle = 0
        if mode == 0:
            mode = 1
            sweep_time = 1.0
            last_movement_at = now
        elif mode == 1:
            mode = 2
            angle = 0
        elif mode == 2:
            mode = 3
```

```
            angle = 180
        elif mode == 3:
            mode = 0
            angle = 0
```

Adjust

If the encoder switch wasn't just pressed, the encoder is use to adjust the relevant parameter (depending on the current mode).

```
    else:
        current_position, change = get_encoder_change(rotary_encoder,
current_position)
        if change != 0:
            if mode == 0:
                angle = min(180, max(0, angle + change * 5))
            elif mode == 1:
                sweep_time = min(5.0, max(1.0, sweep_time + change * 0.1))
            elif mode == 2:
                min_pulse_index = min(10, max(min_pulse_index + change, 0))
                test_servo = servo.Servo(pwm,
                                         min_pulse=min_pulses[min_pulse_index],
                                         max_pulse=max_pulses[max_pulse_index])
                test_servo.angle = 0
            elif mode == 3:
                max_pulse_index = min(10, max(max_pulse_index + change, 0))
                test_servo = servo.Servo(pwm,
                                         min_pulse=min_pulses[min_pulse_index],
                                         max_pulse=max_pulses[max_pulse_index])
                test_servo.angle = 180
```

Display

Once any mode change or adjustment has been made, the display is updated as appropriate for the mode.

```
oled.fill(0)
    if mode == 0:
        oled.text("Angle: {0}".format(angle), 0, 0)
    elif mode == 1:
        oled.text("Sweep time: {0}".format(sweep_time), 0, 0)
    elif mode == 2:
        oled.text("Min width: {0}".format(min_pulses[min_pulse_index]), 0, 0)
    elif mode == 3:
        oled.text("Max width: {0}".format(max_pulses[max_pulse_index]), 0, 0)
    oled.show()
```

Servo

The final step in the loop is to update the angle of the servo.

```
    test_servo.angle = angle
```

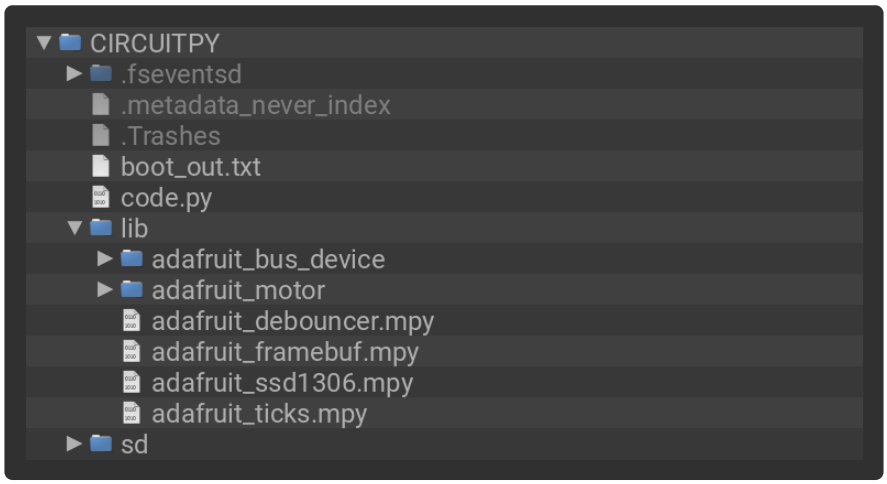
And that's it. Below is the full code. You can download it by clicking "Download Project Bundle." You'll need the debouncer as well which can be found in the library bundle.

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Servo Tester

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Written by Dave Astels for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
Licensed under the MIT license.

All text above must be included in any redistribution.
"""

import time
import board
import busio
import digitalio
import rotaryio
import pwmio
import adafruit_ssd1306
from adafruit_motor import servo
from adafruit_debouncer import Debouncer

#-------------------------------------------------------------------------------
# Initialize Rotary encoder

button_io = digitalio.DigitalInOut(board.D12)
button_io.direction = digitalio.Direction.INPUT
button_io.pull = digitalio.Pull.UP
button = Debouncer(button_io)
rotary_encoder = rotaryio.IncrementalEncoder(board.D10, board.D11)

#-------------------------------------------------------------------------------
# Initialize I2C and OLED

i2c = busio.I2C(board.SCL, board.SDA)

oled = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c)
oled.fill(0)
oled.show()

min_pulses = [ 500,  550,  600,  650,  700,  750,  800,  850,  900,  950, 1000]
max_pulses = [2000, 2050, 2100, 2150, 2200, 2250, 2300, 2350, 2400, 2450, 2500]

min_pulse_index = 10
max_pulse_index = 0

#-------------------------------------------------------------------------------
# Initialize servo

pwm = pwmio.PWMOut(board.D5, frequency=50)
test_servo = servo.Servo(pwm, min_pulse=1000, max_pulse=2000)
test_servo.angle = 0

current_position = None               # current encoder position
change = 0                            # the change in encoder position
angle = 0
```

```
mode = 0
sweep_time = 1.0
last_movement_at = 0.0
delta = 5


def get_encoder_change(encoder, pos):
    new_position = encoder.position
    if pos is None:
        return (new_position, 0)
    else:
        return (new_position, new_position - pos)

#--------------------------------------------------------------------------------
# Main loop

while True:
    now = time.monotonic()
    button.update()

    if mode == 1:
        if now >= (last_movement_at + sweep_time / 36):
            last_movement_at = now
            angle += delta
            if (angle > 180) or (angle < 0):
                delta *= -1
                angle += delta

    if button.fell:
        servo.angle = 0
        if mode == 0:
            mode = 1
            sweep_time = 1.0
            last_movement_at = now
        elif mode == 1:
            mode = 2
            angle = 0
        elif mode == 2:
            mode = 3
            angle = 180
        elif mode == 3:
            mode = 0
            angle = 0

    else:
        current_position, change = get_encoder_change(rotary_encoder,
current_position)
        if change != 0:
            if mode == 0:
                angle = min(180, max(0, angle + change * 5))
            elif mode == 1:
                sweep_time = min(5.0, max(1.0, sweep_time + change * 0.1))
            elif mode == 2:
                min_pulse_index = min(10, max(min_pulse_index + change, 0))
                test_servo = servo.Servo(pwm,
                                         min_pulse=min_pulses[min_pulse_index],
                                         max_pulse=max_pulses[max_pulse_index])
                angle = 0
            elif mode == 3:
                max_pulse_index = min(10, max(max_pulse_index + change, 0))
                test_servo = servo.Servo(pwm,
                                         min_pulse=min_pulses[min_pulse_index],
                                         max_pulse=max_pulses[max_pulse_index])
                angle = 180

    oled.fill(0)
    if mode == 0:
        oled.text("Angle: {0}".format(angle), 0, 0)
    elif mode == 1:
```

```
        oled.text("Sweep time: {0}".format(sweep_time), 0, 0)
    elif mode == 2:
        oled.text("Min width: {0}".format(min_pulses[min_pulse_index]), 0, 0)
    elif mode == 3:
        oled.text("Max width: {0}".format(max_pulses[max_pulse_index]), 0, 0)
    oled.show()

    test_servo.angle = angle
```

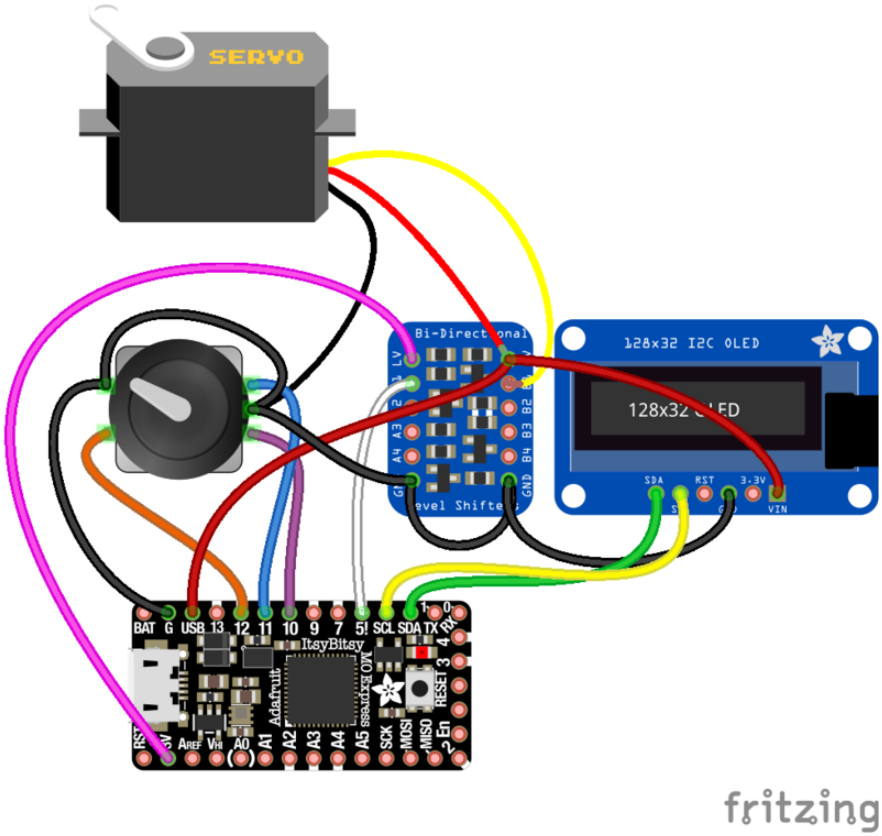
After you've done this, your CIRCUITPY drive should look like this:

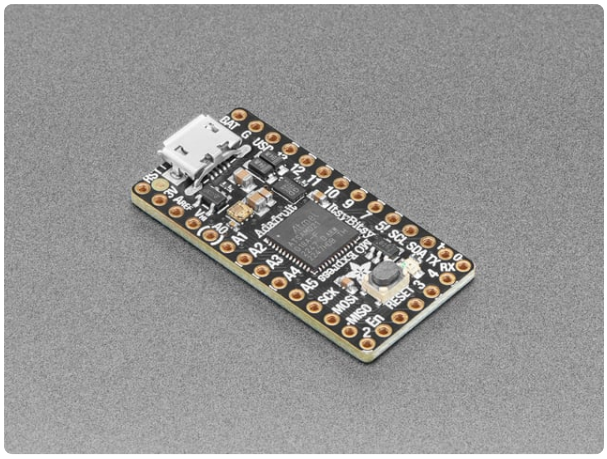


Next Steps

If you do a lot with servos this could be a handy tool to have on your shelf.

If so, the next step might be to simplify it and put it in a case of some sort. The prototype in this guide was built using Feather ecosystem components for ease of construction using the tripler wing. Since using 5v for the servo was a requirement, being battery powered wasn't. That makes it easy to switch to an ItsyBitsy M0 Express (it has no onboard battery support). The OLED display comes in a breakout version that's identical functionally to the FeatherWing for our purpose (it is just the display, no buttons). The level shifter and rotary encoder don't change.





[Adafruit ItsyBitsy M0 Express - for CircuitPython & Arduino IDE]

What's smaller than a Feather but larger than a Trinket? It's an Adafruit ItsyBitsy M0 Express! Small, powerful, with a rockin' ATSAMD21 Cortex M0...

https://www.adafruit.com/product/3727

[Monochrome 128x32 I2C OLED graphic display]

These displays are small, only about 1" diagonal, but very readable due to the high contrast of an OLED display. This display is made of 128x32 individual white OLED pixels, each...

https://www.adafruit.com/product/931

The author anticipates doing this when he gets time to spend in Fusion360, so watch for an update if you are interested.