# Table of Contents

## Overview
- Prerequisite Guides
- Parts
- Cables and Such
- Hardware

## Circuit Diagram
- Adafruit Library for Fritzing
- Wired Connections

## CAD Files
- 3D Printed Parts
- 3D Printed Parts List
- Build Volume
- CAD Assembly

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

## Code the Guitar
- Upload the Code and Libraries to the RP2040 Prop-Maker Feather
- How the CircuitPython Code Works
- Keys and Pixels
- synthio
- The Loop
- Filters
- Keytar Mode

## Wiring
- NeoKey I2C Address
- Daisy Chain NeoKeys
- NeoPixel Stick
- Solder NeoPixel Stick Wires
- Rotary Encoder I2C Addresses
- On/Off Button
- Soldering Ground Wire
- Soldering Common
- Soldering LED Wire
- Soldering NC Pin
- JST Cables for Feather
- Solder Ground
- Solder 3V Pin on Feather
- Solder EN Pin on Feather
- Connect On/Off Button to Feather
- Solder JST plug to Battery Holder
- Testing On/Off Button with Feather
- Wires for SPST Switch
- Solder Wires
- Wires for DPDT Switch
- Soldering wires to DPDT Switch
• Connect DPDT Switch to TRRS
• Solder Wires to TRRS
• Solder Wires to Micro Switches

Assembly

• Hardware for Neck and Head
  • Install NeoKeys to Neck
  • Secure NeoKeys
  • Installed NeoKeys
• Top Body Overlays (Optional)
  • Install Neck to Body
  • Battery Cover
• Connect Speaker Wires to DPDT Switch
• Panel Mount DPDT Switch
• Connect Rotary Encoders
  • Panel Mount Rotary Encoders
• Install SPST Switch
  • Secure SPST Switch
• Connect SPST Switch Ground Wire
  • Secure NeoPixel Stick
  • NeoPixel Mount Hardware
  • Secure NeoPixel Mount
• Micro Switch Hardware
  • Micro Switch Mounts
  • Assembled Micro Switch Mounts
• Strum Hinge Hardware
  • Secure Hinges to Strum Plate
• Install Pegs to Strum Pick
  • Assembled Strum Plate
• Install Hardware for Strum Plate
• Installing Strum Plate
  • Install Strum Pick
  • Secure Strum Assembly
• Disable LiPo Charge on Feather
• Feather Mount Setup
• On/Off Button Hardware
  • Secure On/Off Button
• Checkpoint
  • Secure Feather Mount
  • Secured Feather Mount
• Secure TRRS Breakout
• Cable for Rotary Encoders
  • Rotary Encoder Cable for Feather
• Connect 3-pin PicoBlade cables together
  • Solder SPST Switch Signal Wire to Feather
• Connect STEMMA QT to Feather
  • Solder Micro Switch Ground Wire
• Connect cables from On/Off Button to Feather
• Connect Strum Switches To Feather
• Connect NeoPixel to Feather
• Connect TRRS to Feather
• Install Battery Cover
- Connect Battery to Feather
- Closing Body Halves
- Secure Body Halves
- Kailh Key Switches
- Install Neck Cover
- Install Key Switches
- Install Key Caps
- Final Build

Use

- Volume
- LFO Rate (Tremolo)
- Note Octave
- Strum Mode
- Audio Mode
- Filters
Overview

Build a musical Synth Guitar with Adafruit's RP2040 PropMaker Feather and CircuitPython!

This instrument uses CircuitPython SynthIO to generate musical notes that can be modulated using rotary encoders and the PropMaker Feather's on-board accelerometer.

Use NeoKey 'Cherry MX'-compatible key switches and a guitar-hero-like strum-bar to play notes.

The RP2040 PropMaker Feather features high-quality I2S audio output. You can switch between the on-board speaker or TRRS audio jack.

The NeoPixel change colors depending on the NeoKey that is pressed.

The RP2040 PropMaker's built in speaker driver is not designed for use with headphones or line in.
Prerequisite Guides

For more information on the RP2040 PropMaker Feather and using the SynthIO module in CircuitPython, check out these guides.

- RP2040 PropMaker Documentation ()
- Audio Synthesis with SynthIO ()

Parts

Adafruit RP2040 Prop-Maker Feather with I2S Audio Amplifier
The Adafruit Feather series gives you lots of options for a small, portable, rechargeable microcontroller board. By picking a feather and stacking on a FeatherWing you can create...
https://www.adafruit.com/product/5768

Kailh Mechanical Key Switches - Linear Red - 10 pack
For crafting your very own custom keyboard, these Kailh Red Linear mechanical key switches are deeeee-luxe! With smooth actuation and Cherry MX compatibility,...
https://www.adafruit.com/product/4952
Clear DSA Keycaps for MX Compatible Switches - 10 pack
Get ready to clacky to your heart's content. Here is a 10 pack of clear transparent DSA keycaps for your next mechanical keyboard or https://www.adafruit.com/product/5013

Mono Enclosed Speaker with Plain Wires - 3W 4 Ohm
Listen up! This single 2.8" x 1.2" speaker is the perfect addition to any audio project where you need 4 ohm impedance and 3W or less of power. We... https://www.adafruit.com/product/4445

NeoPixel Stick - 8 x 5050 RGB LED with Integrated Drivers
Make your own little LED strip arrangement with this stick of NeoPixel LEDs. We crammed 8 of the tiny 5050 (5mm x 5mm) smart RGB LEDs onto a PCB with mounting holes and a chainable... https://www.adafruit.com/product/1426

Adafruit TRRS Jack Breakout Board
Tip-Ring-Ring-Sleeve style audio cables are often used for situations where you want stereo audio and then an extra contact for a microphone input. https://www.adafruit.com/product/5764
<table>
<thead>
<tr>
<th>Quantity</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 x</td>
<td><strong>Rotary Encoder Stemma QT</strong>&lt;br&gt;Adafruit I2C Stemma QT Rotary Encoder Breakout with NeoPixel - STEMMA QT / Qwiic</td>
<td><a href="https://www.adafruit.com/product/4991">https://www.adafruit.com/product/4991</a></td>
</tr>
<tr>
<td>1 x</td>
<td><strong>Mini Toggle Switch - SPDT</strong>&lt;br&gt;Mini Toggle Switch - SPDT</td>
<td><a href="https://www.adafruit.com/product/3221">https://www.adafruit.com/product/3221</a></td>
</tr>
<tr>
<td>1 x</td>
<td><strong>Mini Toggle Switch - DPDT</strong>&lt;br&gt;Mini Toggle Switch - DPDT</td>
<td><a href="https://www.adafruit.com/product/3220">https://www.adafruit.com/product/3220</a></td>
</tr>
<tr>
<td>2 x</td>
<td><strong>Micro Switch w/Lever</strong>&lt;br&gt;2 Terminal</td>
<td><a href="https://www.adafruit.com/product/818">https://www.adafruit.com/product/818</a></td>
</tr>
<tr>
<td>1 x</td>
<td><strong>On/Off Switch</strong>&lt;br&gt;Rugged Metal On/Off Switch with Blue LED Ring - 16mm Blue On/Off</td>
<td><a href="https://www.adafruit.com/product/915">https://www.adafruit.com/product/915</a></td>
</tr>
<tr>
<td>1 x</td>
<td><strong>AA Battery Holder</strong>&lt;br&gt;3 x AA Battery Holder with 2.1mm Plug</td>
<td><a href="https://www.adafruit.com/product/3842">https://www.adafruit.com/product/3842</a></td>
</tr>
<tr>
<td>1 x</td>
<td><strong>Alkaline AA batteries</strong>&lt;br&gt;(LR6) - 3 pack</td>
<td><a href="https://www.adafruit.com/product/3521">https://www.adafruit.com/product/3521</a></td>
</tr>
</tbody>
</table>

**Cables and Such**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x</td>
<td><strong>Long STEMMA QT Cable</strong>&lt;br&gt;JST SH 4-Pin Cable - 400mm long</td>
<td><a href="https://www.adafruit.com/product/5385">https://www.adafruit.com/product/5385</a></td>
</tr>
<tr>
<td>1 x</td>
<td><strong>Medium STEMMA QT Cable</strong>&lt;br&gt;STEMMA QT / Qwiic JST SH 4-pin Cable - 100mm Long</td>
<td><a href="https://www.adafruit.com/product/4210">https://www.adafruit.com/product/4210</a></td>
</tr>
<tr>
<td>4 x</td>
<td><strong>Short STEMMA QT Cable</strong>&lt;br&gt;STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long</td>
<td><a href="https://www.adafruit.com/product/4399">https://www.adafruit.com/product/4399</a></td>
</tr>
<tr>
<td>1 x</td>
<td><strong>10-wire Silicone Cover Ribbon Cable</strong>&lt;br&gt;10-wire Silicone Cover Ribbon Cable</td>
<td><a href="https://www.adafruit.com/product/3890">https://www.adafruit.com/product/3890</a></td>
</tr>
<tr>
<td>1 x</td>
<td><strong>1.25mm Pitch 2-pin Cable Matching Pair</strong>&lt;br&gt;40cm long - Molex PicoBlade Compatible</td>
<td><a href="https://www.adafruit.com/product/4720">https://www.adafruit.com/product/4720</a></td>
</tr>
<tr>
<td>1 x</td>
<td><strong>JST PH 2-Pin Cable</strong>&lt;br&gt;Male Header 200mm</td>
<td><a href="https://www.adafruit.com/product/3814">https://www.adafruit.com/product/3814</a></td>
</tr>
<tr>
<td>2 x</td>
<td><strong>JST PH 2-Pin Cable</strong></td>
<td></td>
</tr>
</tbody>
</table>
Female Connector 100mm

https://www.adafruit.com/product/261

1 x 1.25mm Pitch 3-pin Cable Matching Pair
40cm long Molex PicoBlade Compatible

https://www.adafruit.com/product/4721

1 x Clear Adhesive Squares
6 pack - UGlu Dashes

https://www.adafruit.com/product/4813

Hardware

- 4x M3 x 16mm button head screws
- 30x M3 x 10mm button head screws
- 2x M3 x 6mm button head screws
- 2x M2.5 x 6mm button head screws
- 8x M2.5 x 8mm button head screws
- 4x M2 x 8mm button head screws
- 30x M3 hex nuts

Circuit Diagram

The diagram below provides a general visual reference for wiring of the components once you get to the Assembly page. This diagram was created using the software package Fritzing.

Adafruit Library for Fritzing

Adafruit uses the Adafruit's Fritzing parts library to create circuit diagrams for projects. You can download the library or just grab individual parts. Get the library and parts from GitHub - Adafruit Fritzing Parts.
Wired Connections

The RP2040 PropMaker Feather is powered by a 3x AA battery pack.

- The NeoPixel stick is connected to pins on the screw block terminal.
- The speaker is connected to the DPDT toggle switch.
- The TRRS breakout is connected to the DPDT toggle switch.
- The SPST toggle is connected to D12 on Feather
- The strum switches are connected to the screw block terminal.
- The on/off switch is connected to EN pin on Feather.
- The rotary encoders and neokeys are chained together with stemma QT cables.

CAD Files

3D Printed Parts

STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material using PLA filament. Original design source may be downloaded using the links below.
3D Printed Parts List

- Battery Cover.stl
- Bottom Body.stl
- Top Body.stl
- Head Top.stl
- Head Bottom.stl
- Neck Cover.stl
- Neck Case.stl
- 2x Strum-Hinge-Pin.stl
- 2x Strum-Hinge.stl
- Strum-Switch-Mount-B.stl
- Strum-Switch-Mount-A.stl
- Strum-Plate.stl
- Strum-Pick.stl
- Feather Mount.stl
- Pixel Mount.stl
- Button Washer.stl
- Audio Overlay.stl
- Strum Overlay.stl
- Body Overlay.stl

Build Volume
The parts require a 3D printer with a minimum build volume.

228mm (X) x 184mm (Y) x 21mm (Z)
CAD Assembly
The strum mechanism is secured to the bottom half of the guitar. The on/off button, rotary encoders and toggle switches are panel mounted to the top half of the guitar. The neck is secured to the top half of the guitar. The head stock is secured to the neck. The neck and head stock covers snap fit. The speaker is secured to the head stock cover. The key switches are press fitted into the neck cover.

CircuitPython

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

CircuitPython Quickstart

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

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

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

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

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

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

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

Drag the adafruit_circuitpython_etc.uf2 file to RPI-RP2.

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

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

Safe Mode

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

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

Entering Safe Mode

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

In Safe Mode

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

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

```
Auto-reload is off.
Running in safe mode! Not running saved code.

CircuitPython is in safe mode because you pressed the reset button during boot.
Press again to exit safe mode.

Press any key to enter the REPL. Use CTRL-D to reload.
```

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

Flash Resetting UF2

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

Download flash erasing "nuke" UF2

Code the Guitar

Once you've finished setting up your RP2040 Prop-Maker Feather 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.

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

import digitalio
import audiobusio
import board
import neopixel
import adafruit_lis3dh
import synthio
import keypad
from adafruit_ticks import ticks_ms, ticks_add, ticks_diff
import ulab.numpy as np
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.neopixel import NeoPixel as SS_NeoPixel
from adafruit_seesaw.digitalio import DigitalIO
from adafruit_seesaw.rotaryio import IncrementalEncoder
import audiomixer
import busio
import simpleio

i2c = busio.I2C(board.SCL, board.SDA, frequency=800000)

int1 = digitalio.DigitalInOut(board.ACCELEROMETER_INTERRUPT)
lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c, int1=int1)
lis3dh.range = adafruit_lis3dh.RANGE_2_G

ss_enc0 = Seesaw(i2c, addr=0x36)
ss_enc0.pin_mode(24, ss_enc0.INPUT_PULLUP)
button0 = DigitalIO(ss_enc0, 24)
button0_state = False
enc0 = IncrementalEncoder(ss_enc0)
ss_enc0.set_GPIO_interrupts(1 << 24, True)
ss_enc0.enable_encoder_interrupt(encoder=0)

ss_enc1 = Seesaw(i2c, addr=0x37)
ss_enc1.pin_mode(24, ss_enc1.INPUT_PULLUP)
button1 = DigitalIO(ss_enc1, 24)
button1_state = False
enc1 = IncrementalEncoder(ss_enc1)
ss_enc1.set_GPIO_interrupts(1 << 24, True)
ss_enc1.enable_encoder_interrupt(encoder=0)

ss_enc2 = Seesaw(i2c, addr=0x38)
ss_enc2.pin_mode(24, ss_enc2.INPUT_PULLUP)
```
button2 = DigitalIO(ss_enc2, 24)
button2_state = False
enc2 = IncrementalEncoder(ss_enc2)
ss_enc2.set_GPIO_interrupts(1 << 24, True)
ss_enc2.enable_encoder_interrupt(encoder=0)

neokey0 = Seesaw(i2c, addr=0x30)
neokey1 = Seesaw(i2c, addr=0x31)

keys = []
for k in range(4, 8):
    key0 = DigitalIO(neokey0, k)
    key0.direction = digitalio.Direction.INPUT
    key0.pull = digitalio.Pull.UP
    keys.append(key0)
for k in range(4, 8):
    key1 = DigitalIO(neokey1, k)
    key1.direction = digitalio.Direction.INPUT
    key1.pull = digitalio.Pull.UP
    keys.append(key1)

NUM_PIXELS = 8
NEOPIXEL_PIN = board.EXTERNAL_NEOPIXELS

pixels = neopixel.NeoPixel(NEOPIXEL_PIN, NUM_PIXELS, auto_write=True)
pixels.brightness = 0.1

enable = digitalio.DigitalInOut(board.EXTERNAL_POWER)
enable.direction = digitalio.Direction.OUTPUT
enable.value = True
strum_switch = digitalio.DigitalInOut(board.D12)
strum_switch.direction = digitalio.Direction.INPUT
strum_switch.pull = digitalio.Pull.UP

int_keys = keypad.Keys((board.D5, board.D6, board.D9, board.EXTERNAL_BUTTON),
    value_when_pressed=False, pull=True, interval = 0.001)

key_pix0 = SS_NeoPixel(neokey0, 3, 4, auto_write = True)
key_pix0.brightness = 1
key_pix1 = SS_NeoPixel(neokey1, 3, 4, auto_write = True)
key_pix1.brightness = 1

key_pixels = [key_pix0[0], key_pix0[1], key_pix0[2], key_pix0[3],
              key_pix1[0], key_pix1[1], key_pix1[2], key_pix1[3]]

# i2s audio
audio = audiobusio.I2SOut(board.I2S_BIT_CLOCK, board.I2S_WORD_SELECT,
    board.I2S_DATA)

key_states = [False, False, False, False, False, False, False, False]
key_colors = [0xFF0000, 0xFF5500, 0xFFFF00, 0x00FF00, 0x00FFFF, 0x0000FF, 0x5500FF,
              0xFF00FF]
for c in range(4):
    key_pix0[c] = key_colors[c]
    key_pix1[c] = key_colors[c + 4]

SAMPLE_RATE = 22050
SAMPLE_SIZE = 512
VOLUME = 32000

square = np.concatenate((np.ones(SAMPLE_SIZE//2,
    dtype=np.int16)*VOLUME,np.ones(SAMPLE_SIZE//2,
    dtype=np.int16)*-VOLUME))
sine = np.array(np.sin(np.linspace(0, 2*np.pi, SAMPLE_SIZE, endpoint=False)) *
    VOLUME,
    dtype=np.int16)

amp_env = synthio.Envelope(attack_time=0.01,
```python
sustain_level=0.5,
release_time=0.1)

lfo_tremo = synthio.LFO(waveform=sine, rate=5)
synth = synthio.Synthesizer(sample_rate=SAMPLE_RATE)
synth_notes = []
octave = 12
mult = 2
octave_range = 6
tones = [36, 40, 43, 47, 50, 53, 57, 60]
diatomic = 0
t = [0, 0, 0, 0, 0, 0, 0, 0]
current_freq = []
for s in range(8):
    t[s] = tones[s] + (octave * mult)
    print(t[s])
    note = synthio.Note(frequency=synthio.midi_to_hz(t[s]),
                        envelope=amp_env, waveform=square,
                        amplitude=lfo_tremo)
    synth_notes.append(note)
current_freq.append(synth_notes[s].frequency)
lfo_frequency = 2000
lfo_resonance = 1.5
lpf = synth.low_pass_filter(lfo_frequency, lfo_resonance)
hpf = synth.high_pass_filter(lfo_frequency, lfo_resonance)
synth_volume = 0.3
last_pos0 = synth_volume
last_pos1 = 0
last_pos2 = 0
mixer = audiomixer.Mixer(voice_count=1, sample_rate=SAMPLE_RATE, channel_count=1,
bits_per_sample=16, samples_signed=True, buffer_size=2048)
audio.play(mixer)
mixer.voice[0].play(synth)
mixer.voice[0].level = synth_volume
int_number = 0
def normalize(val, min_v, max_v):
    return max(min(max_v, val), min_v)
key_pressed = 0
strum = False
last_strum = strum
tremolo = True
pressed_notes = []
last_y = 0
accel_time = 0.1
accel_clock = ticks_ms()
accel_time = int(accel_time * 1000)
while True:
    interrupt_event = int_keys.events.get()
    strum = strum_switch.value
    if last_strum != strum:
        synth.release_all()
        last_strum = strum
    if interrupt_event:
        int_number = interrupt_event.key_number
        if int_number == 0 and interrupt_event.pressed:
            pos0 = -enc0.position
            if pos0 != last_pos0:
                if pos0 > last_pos0:
                    synth_volume = synth_volume + 0.1
```

©Adafruit Industries
else:
    synth_volume = synth_volume - 0.1
    synth_volume = normalize(synth_volume, 0.0, 1.0)
    print(synth_volume)
mixer.voice[0].level = synth_volume
last_pos0 = pos0
if not button0.value and not button0_state:
    button0_state = True
    if mixer.voice[0].level > 0:
        mixer.voice[0].level = 0
    else:
        mixer.voice[0].level = synth_volume
if button0.value and button0_state:
    button0_state = False
elif int_number == 1 and interrupt_event.pressed:
    pos1 = -enc1.position
    if pos1 != last_pos1:
        if pos1 > last_pos1:
            lfo_tremo.rate = lfo_tremo.rate + 0.5
        else:
            lfo_tremo.rate = lfo_tremo.rate - 0.5
        lfo_tremo.rate = normalize(lfo_tremo.rate, 1.0, 20.0)
        print(lfo_tremo.rate)
        last_pos1 = pos1
if tremolo:
    if not button1.value and not button1_state:
        button1_state = True
        tremolo = False
    for i in range(8):
        synth_notes[i].amplitude = 1.0
    if button1.value and button1_state:
        button1_state = False
    else:
        if not button1.value and not button1_state:
            button1_state = True
            tremolo = True
        for i in range(8):
            lfo_tremo.rate = 5.0
            synth_notes[i].amplitude = lfo_tremo
        if button1.value and button1_state:
            button1_state = False
elif int_number == 2 and interrupt_event.pressed:
    pos2 = -enc2.position
    if pos2 != last_pos2:
        if pos2 > last_pos2:
            mult = (mult + 1) % octave_range
            print(mult)
            for o in range(8):
                t[o] = tones[o] + (octave * mult)
                print(t[o])
                synth_notes[o].frequency = synthio.midi_to_hz(t[o])
                current_freq[o] = synth_notes[o].frequency
        else:
            mult = (mult - 1) % octave_range
            print(mult)
            for o in range(8):
                t[o] = tones[o] + (octave * mult)
                print(t[o])
                synth_notes[o].frequency = synthio.midi_to_hz(t[o])
                current_freq[o] = synth_notes[o].frequency
        last_pos2 = pos2
if not button2.value and not button2_state:
    button2_state = True
    diatonic = (diatonic + 1) % 2
    print(diatonic)
    if diatonic == 0:
        new_tones = [36, 40, 43, 47, 50, 53, 57, 60]
        for r in range(8):
            tones[r] = new_tones[r]
print(tones[r])
else:
    new_tones = [36, 38, 40, 41, 43, 45, 47, 48]
    for r in range(8):
        tones[r] = new_tones[r]
        print(tones[r])
    for x in range(8):
        t[x] = tones[x] + (octave * mult)
        print(t[x])
        synth_notes[x].frequency = synthio.midi_to_hz(t[x])
        current_freq[x] = synth_notes[x].frequency
    if button2.value and button2_state:
        button2_state = False
    elif int_number == 3 and interrupt_event.pressed:
        if strum:
            for i in range(0, 8):
                if not keys[i].value:
                    pixels.fill(key_colors[i])
                    pixels.show()
        synth_notes[i] = None
    elif int_number == 3 and interrupt_event.released:
        if strum:
            synth.release_all()
            ss_enc0.get_GPIO_interrupt_flag()
            ss_enc1.get_GPIO_interrupt_flag()
            ss_enc2.get_GPIO_interrupt_flag()
            if ticks_diff(ticks_ms(), accel_clock) >= accel_time:
                x, y, z = [value / adafruit_lis3dh.STANDARD_GRAVITY for value in lis3dh.acceleration]
                if last_y != y:
                    if abs(last_y - y) > 0.01:
                        # print(f"x = {x:.3f} G, y = {y:.3f} G, z = {z:.3f} G")
                        if y < -0.500:
                            mapped_freq = simpleio.map_range(y, -0.300, -1, 2000, 10000)
                            mapped_resonance = simpleio.map_range(y, -0.300, -1, 1.5, 8)
                            hpf = synth.high_pass_filter(mapped_freq, mapped_resonance)
                            for i in range(0, 8):
                                synth_notes[i].filter = hpf
                            elf y > 0.200:
                                mapped_freq = simpleio.map_range(y, 0.200, 1, 2000, 100)
                                mapped_resonance = simpleio.map_range(y, 0.200, 1, 2, 0.5)
                                lpf = synth.low_pass_filter(mapped_freq, mapped_resonance)
                                for i in range(0, 8):
                                    synth_notes[i].filter = lpf
                        else:
                            for i in range(0, 8):
                                synth_notes[i].filter = None
                        last_y = y
                    accel_clock = ticks_add(accel_clock, accel_time)
                    if not strum:
                        for i in range(0, 8):
                            if not keys[i].value and not key_states[i]:
                                pixels.fill(key_colors[i])
                                pixels.show()
                                synth_notes[i] = True
                                if not keys[i].value and key_states[i]:
                                    key_pixels[i] = key_colors[i]
                                    synth.release(synth_notes[i])
                                    key_states[i] = False
                else:
                    if keys[i].value and key_states[i]:
                        key_pixels[i] = key_colors[i]
Upload the Code and Libraries to the RP2040 Prop-Maker Feather

After downloading the Project Bundle, plug your RP2040 Prop-Maker Feather 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 RP2040 Prop-Maker Feather's CIRCUITPY drive.

- lib folder
- code.py

Your RP2040 Prop-Maker Feather CIRCUITPY drive should look like this after copying the lib folder and the code.py file.

How the CircuitPython Code Works

The code begins by instantiating the onboard accelerometer, three seesaw rotary encoders and both 1x4 NeoKeys over I2C. The rotary encoders are using their interrupt pins.

```python
i2c = busio.I2C(board.SCL, board.SDA, frequency=800000)
int1 = digitalio.DigitalInOut(board.ACCELEROMETER_INTERRUPT)
lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c, int1=int1)
lis3dh.range = adafruit_lis3dh.RANGE_2_G

ss_enc0 = Seesaw(i2c, addr=0x36)
ss_enc0.pin_mode(24, ss_enc0.INPUT_PULLUP)
button0 = DigitalIO(ss_enc0, 24)
button0_state = False
enc0 = IncrementalEncoder(ss_enc0)
ss_enc0.set_GPIO_interrupts(1 << 24, True)
ss_enc0.enable_encoder_interrupt(encoder=0)
```
ss_enc1 = Seesaw(i2c, addr=0x37)
ss_enc1.pin_mode(24, ss_enc1.INPUT_PULLUP)
button1 = DigitalIO(ss_enc1, 24)
button1_state = False
enc1 = IncrementalEncoder(ss_enc1)
ss_enc1.set_GPIO_interrupts(1 &lt;&lt; 24, True)
ss_enc1.enable_encoder_interrupt(encoder=0)

ss_enc2 = Seesaw(i2c, addr=0x38)
ss_enc2.pin_mode(24, ss_enc2.INPUT_PULLUP)
button2 = DigitalIO(ss_enc2, 24)
button2_state = False
enc2 = IncrementalEncoder(ss_enc2)
ss_enc2.set_GPIO_interrupts(1 &lt;&lt; 24, True)
ss_enc2.enable_encoder_interrupt(encoder=0)

neokey0 = Seesaw(i2c, addr=0x30)
neokey1 = Seesaw(i2c, addr=0x31)

Keys and Pixels

The individual keys and NeoPixels on the NeoKey breakouts are created.

```python
keys = []
for k in range(4, 8):
    key0 = DigitalIO(neokey0, k)
    key0.direction = digitalio.Direction.INPUT
    key0.pull = digitalio.Pull.UP
    keys.append(key0)
for k in range(4, 8):
    key1 = DigitalIO(neokey1, k)
    key1.direction = digitalio.Direction.INPUT
    key1.pull = digitalio.Pull.UP
    keys.append(key1)

key_pix0 = SS_NeoPixel(neokey0, 3, 4, auto_write = True)
key_pix0.brightness = 1
key_pix1 = SS_NeoPixel(neokey1, 3, 4, auto_write = True)
key_pix1.brightness = 1

key_pixels = [key_pix0[0], key_pix0[1], key_pix0[2], key_pix0[3],
              key_pix1[0], key_pix1[1], key_pix1[2], key_pix1[3]]
key_states = [False, False, False, False, False, False, False, False]
key_colors = [0xFF0000, 0xFF5500, 0xFFFF00, 0x00FF00, 0x00FFFF, 0x0000FF, 0x5500FF,
              0xFF00FF]
for c in range(4):
    key_pix0[c] = key_colors[c]
    key_pix1[c] = key_colors[c + 4]
```

synthio

This guitar uses synthio to make music. Audio is output via the onboard I2S amp on the Feather. The note frequencies are passed as MIDI note numbers so that its mathematically easy to increase or decrease octaves in the loop.
audio = audiobusio.I2SOut(board.I2S_BIT_CLOCK, board.I2S_WORD_SELECT, board.I2S_DATA)
SAMPLE_RATE = 22050
SAMPLE_SIZE = 512
VOLUME = 32000

square = np.concatenate((np.ones(SAMPLE_SIZE//2, dtype=np.int16)*VOLUME, np.ones(SAMPLE_SIZE//2, dtype=np.int16)*-VOLUME))
sine = np.array(np.sin(np.linspace(0, 2*np.pi, SAMPLE_SIZE, endpoint=False)) * VOLUME, dtype=np.int16)

amp_env = synthio.Envelope(attack_time=0.01, sustain_level=0.5, release_time=0.1)
lfo_tremo = synthio.LFO(waveform=sine, rate=5)
synth = synthio.Synthesizer(sample_rate=SAMPLE_RATE)
synth_notes = []
octave = 12
mult = 2
octave_range = 6
tones = [36, 40, 43, 47, 50, 53, 57, 60]
diatonic = 0
t = [0, 0, 0, 0, 0, 0, 0, 0]
current_freq = []
for s in range(8):
t[s] = tones[s] + (octave * mult)
print(t[s])
note = synthio.Note(frequency=synthio.midi_to_hz(t[s]), envelope=amp_env, waveform=square, amplitude = lfo_tremo)
synth_notes.append(note)
current_freq.append(synth_notes[s].frequency)
lfo_frequency = 2000
lfo_resonance = 1.5
lpf = synth.low_pass_filter(lfo_frequency, lfo_resonance)
hpf = synth.high_pass_filter(lfo_frequency, lfo_resonance)

The audio output is passed thru a Mixer object, allowing for volume control through software.

synth_volume = 0.3
last_pos0 = synth_volume
last_pos1 = 0
last_pos2 = 0
mixer = audiomixer.Mixer(voice_count=1, sample_rate=SAMPLE_RATE, channel_count=1, bits_per_sample=16, samples_signed=True, buffer_size=2048)
audio.play(mixer)
mixer.voice[0].play(synth)
mixer.voice[0].level = synth_volume
The Loop

In the loop, the interrupt pins and strummer pin are scanned for any new events. Encoder 0 controls the volume of the synth. If you press the encoder switch it will mute the synth.

```python
if interrupt_event:
    int_number = interrupt_event.key_number
    if int_number == 0 and interrupt_event.pressed:
        pos0 = -enc0.position
        if pos0 != last_pos0:
            if pos0 &gt; last_pos0:
                synth_volume = synth_volume + 0.1
            else:
                synth_volume = synth_volume - 0.1
        synth_volume = normalize(synth_volume, 0.0, 1.0)
        print(synth_volume)
    mixer.voice[0].level = synth_volume
    last_pos0 = pos0
    if not button0.value and not button0_state:
        button0_state = True
    if mixer.voice[0].level &gt; 0:
        mixer.voice[0].level = 0
    else:
        mixer.voice[0].level = synth_volume
    if button0.value and button0_state:
        button0_state = False
```

Encoder 1 controls the LFO rate. If you press the encoder button it enables or disables the LFO.

```python
elif int_number == 1 and interrupt_event.pressed:
    pos1 = -enc1.position
    if pos1 != last_pos1:
        if pos1 &gt; last_pos1:
            lfo_tremo.rate = lfo_tremo.rate + 0.5
        else:
            lfo_tremo.rate = lfo_tremo.rate - 0.5
        lfo_tremo.rate = normalize(lfo_tremo.rate, 1.0, 20.0)
        print(lfo_tremo.rate)
    last_pos1 = pos1
    if tremolo:
        if not button1.value and not button1_state:
            button1_state = True
            tremolo = False
        for i in range(8):
            synth_notes[i].amplitude = 1.0
        if button1.value and button1_state:
            button1_state = False
    else:
        if not button1.value and not button1_state:
            button1_state = True
        for i in range(8):
            lfo_tremo.rate = 5.0
            synth_notes[i].amplitude = lfo_tremo
        if button1.value and button1_state:
            button1_state = False
```
Encoder 2 controls the note frequencies. Turning the encoder increases or decreases the octave range of the guitar. Pressing the encoder button switches between triad or diatonic mode.

```python
elif int_number == 2 and interrupt_event.pressed:
    pos2 = -enc2.position
    if pos2 != last_pos2:
        if pos2 > last_pos2:
            mult = (mult + 1) % octave_range
            print(mult)
            for o in range(8):
                t[o] = tones[o] + (octave * mult)
                synth_notes[o].frequency = synthio.midi_to_hz(t[o])
                current_freq[o] = synth_notes[o].frequency
        else:
            mult = (mult - 1) % octave_range
            print(mult)
            for o in range(8):
                t[o] = tones[o] + (octave * mult)
                synth_notes[o].frequency = synthio.midi_to_hz(t[o])
                current_freq[o] = synth_notes[o].frequency
    last_pos2 = pos2
    if not button2.value and not button2_state:
        button2_state = True
        diatonic = (diatonic + 1) % 2
        print(diatonic)
        if diatonic == 0:
            new_tones = [36, 40, 43, 47, 50, 53, 57, 60]
            for r in range(8):
                tones[r] = new_tones[r]
                print(tones[r])
        else:
            new_tones = [36, 38, 40, 41, 43, 45, 47, 48]
            for r in range(8):
                tones[r] = new_tones[r]
                print(tones[r])
            for x in range(8):
                t[x] = tones[x] + (octave * mult)
                print(t[x])
                synth_notes[x].frequency = synthio.midi_to_hz(t[x])
                current_freq[x] = synth_notes[x].frequency
    if button2.value and button2_state:
        button2_state = False
```

The final Keypad pin is for the strum bar. If strum mode is enabled, it plays the currently pressed synth note. After scanning, the GPIO interrupt flags are called for the three encoders. This resets the interrupt pins.

```python
elif int_number == 3 and interrupt_event.pressed:
    if strum:
        for i in range(0, 8):
            if not keys[i].value:
                pixels.fill(key_colors[i])
                pixels.show()
                synth.press(synth_notes[i])
elif int_number == 3 and interrupt_event.released:
    if strum:
        synth.release_all()
        ss_enc0.get_GPIO_interrupt_flag()
```

©Adafruit Industries
Filters

The accelerometer controls whether a high pass or low pass filter is applied to the synth. If the guitar is tilted up, a high pass filter is enabled. If the guitar is tilted down, a low pass filter is enabled. The filter frequency and resonance are affected by the readings from the accelerometer.

```python
if ticks_diff(ticks_ms(), accel_clock) >= accel_time:
    x, y, z = [value / adafruit_lis3dh.STANDARD_GRAVITY for value in lis3dh.acceleration]
    if last_y != y:
        if abs(last_y - y) > 0.01:
            # print(f"x = {x:.3f} G, y = {y:.3f} G, z = {z:.3f} G")
            if y < -0.500:
                mapped_freq = simpleio.map_range(y, -0.300, -1, 2000, 10000)
                mapped_resonance = simpleio.map_range(y, -0.300, -1, 1.5, 8)
                hpf = synth.high_pass_filter(mapped_freq, mapped_resonance)
                for i in range(0, 8):
                    synth_notes[i].filter = hpf
            elif y > 0.200:
                mapped_freq = simpleio.map_range(y, 0.200, 1, 2000, 100)
                mapped_resonance = simpleio.map_range(y, 0.200, 1, 2, 0.5)
                lpf = synth.low_pass_filter(mapped_freq, mapped_resonance)
                for i in range(0, 8):
                    synth_notes[i].filter = lpf
            else:
                for i in range(0, 8):
                    synth_notes[i].filter = None
        last_y = y
    accel_clock = ticks_add(accel_clock, accel_time)
```

Keytar Mode

If `strum` mode is not enabled, then pressing the keys will play the synth.

```python
if not strum:
    for i in range(0, 8):
        if not keys[i].value and not key_states[i]:
            pixels.fill(key_colors[i])
            pixels.show()
            synth.press(synth_notes[i])
            key_states[i] = True
        if keys[i].value and key_states[i]:
            key_pixels[i] = key_colors[i]
            synth.release(synth_notes[i])
            key_states[i] = False
```
Wiring

**NeoKey I2C Address**
One of the NeoKeys will need an I2C address.

Bridge the A0 pads on the NeoKey PCB using a blob of solder.

**Daisy Chain NeoKeys**
Use a short STEMMA QT cable to connect the two NeoKey PCBs together.

Connect a longer STEMMA QT cable to the NeoKey with the A0 I2C address.

**NeoPixel Stick**
Use the ribbon cable to create a 3-pin wire that is 8 inches / 20.3 cm in length.
Solder NeoPixel Stick Wires
Connect the three wires to the GND, DIN and 5VDC pads on the NeoPixel stick.

Rotary Encoder I2C Addresses
Two of the three rotary encoders will need to be assigned special I2C addresses by bridging the jumpers with a blob of solder.

1st Rotary Encoder no I2C Address
2nd Rotary Encoder A1 I2C Address
3rd Rotary Encoder A0 I2C Address
On/Off Button
Use the 2-pin JST PH and 2-pin PicoBlade socket cables and a short (1 inch long) wire.

Soldering Ground Wire
Connect the short wire and black wire from the 2-pin PicoBlade socket cable to the cathode (− LED) terminal.

Soldering Common
Connect the short wire and black wire from the 2-pin JST PH socket cable to the common (C1) terminal.
Soldering LED Wire
Connect the red wire from the 2-pin PicoBlade socket cable to the anode (+ LED) terminal.

Soldering NC Pin
Connect the red wire from the 2-pin JST PH socket cable to the normal closed (NC1) terminal.

JST Cables for Feather
Use the 2-pin JST PH and 2-pin PicoBlade plug cables on the Feather.
Solder Ground
Connect the two black wires to the ground pin.

Solder 3V Pin on Feather
Connect the red wire from the PicoBlade cable to the 3V pin.

Solder EN Pin on Feather
Connect the red wire from the JST PH cable to the EN pin.
Connect On/Off Button to Feather
Plug in the corresponding cables together.

Solder JST plug to Battery Holder
Cut the stock 2.1mm barrel jack from the 3x AA battery holder.
Solder the 2-pin JST female cable to the power cable.
Ensure the polarity of the wires are correct.

Ensure the wire polarities are correct before powering!

Testing On/Off Button with Feather
Install AA batteries into the battery holder.
Plug the battery holder into the Feather.
Use the On/Off Button to power the Feather to test the circuit.
Wires for SPST Switch
Use the ribbon cable to create two wires.

Short Wire 2.5in / 6.35 cm long
Longer Wire 8in / 20.3 cm long

Solder Wires
Attach the short wire to one of the terminals (not the middle).

Attach the longer wire to the middle terminal.
Wires for DPDT Switch
Use ribbon cable to create two sets of 2-pin wires that are 8 inches / 20.3 cm in length.

Soldering wires to DPDT Switch
Attach the first 2-pin cable to one of the terminals that isn't the middle.

Attach the second 2-pin cable to the middle terminal.
Connect DPDT Switch to TRRS
The wires from the DPDT switch will be soldered to the pins on the TRRS breakout.

Solder Wires to TRRS
Attach one wire from the 2-pin cable to the Left labeled pin.
Attach one wire from the other 2-pin cable to the Ring labeled pin.

Solder Wires to Micro Switches
Use the ribbon cable to great four wires.
2x Long Wires 6.5 inches / 16.5 cm
2x Short Wires 2.5 inches / 6.35 cm
Use the short wires to share the COM and NO connections across the two switches.
Connect the longer wires to the switches COM and NO terminals.
Reference the photo for correct wiring.
Assembly

Hardware for Neck and Head
Use the following hardware to secure the neck case to the head case.

2x M3 x 10mm long screw
2x M3 hex nuts
Bring the parts together and line up the mounting holes. Hold the parts together while inserting the M3 screw through the mounting holes. Use hex nuts to secure the parts together.

Hardware for Speaker
Use the following hardware to secure the speaker to the cover.

4x M3 x 10mm long screws
4x M3 hex nuts
Orient the speaker with the cover so the cable is in the proper location.
Secure Speaker
Place the speaker over the cover and line up the mounting holes.

Insert the M3 screws through the cover and speaker mounting holes. Use the speaker using the M3 hex nuts.
Install Speaker Cover
Orient the speaker cover with the head case so the parts match.

Firmly press the speaker cover into the head case.

Install NeoKeys to Neck
Orient the two NeoKey PCBs with the neck assembly.

Use the following hardware to secure the NeoKeys.
8x M2.5 x 8mm long screws
Place the speaker cable in between the built-in standoffs and exiting the end of the neck.
Secure NeoKeys
Place the two NeoKey PCBs into the built-in standoffs on the neck case.

Insert and fasten the M2.5 screws to secure the NeoKey PCBs.

Installed NeoKeys
Ensure the speaker cable and STEMMA QT cable are going through the notch on the end of the neck.
Top Body Overlays (Optional)
Attach the overlay panels to the top half of the guitar body using double-sided sticky tape or glue.

Install Neck to Body
Use the following hardware to secure the neck to the top half of the guitar body.

- 2x M3 x 10mm long screws
- 2x M3 hex nuts
Secure Neck to Body
Insert the M3 screws through the mounting tabs on the top half of the guitar body.

Use the M3 hex nuts to secure the neck to the body.
Battery Cover
Orient the 3x AA battery holder with the battery cover.

Attach the 3x AA battery holder to the battery cover using the clear adhesive squares.
Connect Speaker Wires to DPDT Switch
Solder the two wires from the speaker to the remaining terminals on the DPDT switch.
Panel Mount DPDT Switch
Fit the DPDT switch through the corresponding mounting hole in the top half of the guitar body.

Use the included hex nut and washer to secure the switch.

Connect Rotary Encoders
Orient the three rotary encoders in the correct order and connect them together using the short STEMMA QT cables.

Connect the STEMMA QT cable from the NeoKeys to the corresponding rotary encoder.

Connect the longer STEMMA QT cable to the corresponding rotary encoder.
Panel Mount Rotary Encoders
Insert the three rotary encoders to the corresponding mounting holes in the top half of the guitar body.

Use the included hex nuts to secure the rotary encoders.

Install SPST Switch
Get the SPST toggle switch ready to panel mount into the top half of the guitar body.

Secure SPST Switch
Use the included hex nut and washer to secure the SPST switch to the top half of the guitar body.
Connect SPST Switch
Ground Wire
Solder the ground wire from the SPST switch to the ground pin on one of the rotary encoder PCBs.

Secure NeoPixel Stick
Place the NeoPixel stick onto the mount and line up the mounting holes.
Secure the 8x NeoPixel stick to the NeoPixel mount using two M2 x 8mm long screws.
NeoPixel Mount Hardware
Use the following hardware to secure the NeoPixel mount to the bottom half of the guitar body.

2x M3 x 10mm long screws
2x M3 hex nuts

Secure NeoPixel Mount
Fasten the two M3 screws through the corresponding mounting holes on the bottom half of the guitar body.

Orient the mounting holes on the NeoPixel mount with the M3 screws.

Secure the mount using the hex nuts.
Micro Switch Hardware
Use the following hardware to secure the micro switches to the switch mounts.

- 4x M3 x 16mm long screws
- 4x M3 hex nuts

Micro Switch Mounts
Place the micro switch onto the mount with the mounting holes lined up.

- Use the hex nuts to secure the micro switch.
- Repeat this process for the second micro switch.
Assembled Micro Switch Mounts
Reference the photo for orienting the two micro switches.

Strum Hinge Hardware
Use the following hardware to secure the two hinges to the strum plate.

4x M3 x 10mm long screws
4x M3 hex nuts
Secure Hinges to Strum Plate

Insert an M3 screw through the countersunk standoffs and place the hinge over the screw. Use the hex nut to secure the hinge in place.

Repeat this process for the second hinge.
Install Pegs to Strum Pick
Insert a peg through the side hole on the strum pick. Use a glue adhesive to permanently bond the peg.

Repeat this process for the second peg.

Assembled Strum Plate
Allow the glue to dry before proceeding with the assembly.
Install Hardware for Strum Plate
Fasten the four M3 x 10mm through the mounting holes for the strum plate on the bottom half of the guitar body.

Installing Strum Plate
Place the strum plate assembly over the M3 screws so they are fitted through the mounting holes.

Place the two micro switch mounts over the strum plate so the M3 screws are fitted through the mounting holes.
Install Strum Pick
Slightly flex the hinges apart so that the pegs of the strum pick can be fitted into the hinges.

Secure Strum Assembly
Insert and fasten the four M3 hex nuts onto the M3 screws to secure the strum assembly to the bottom half of the guitar body.

Disable LiPo Charge on Feather
Locate the LiPo charge trace on the back of the RP2040 PropMaker Feather.

Use a hobby knife to cut the trace to disable LiPo charging.
Feather Mount Setup

Use 2x M2.5 x 6mm long screws to secure the Feather PCB to the feather mount.

Place the Feather onto the mount. Insert and fasten the screws to secure the feather.
On/Off Button Hardware
Get the on/off button, hex nut and 3D printed washer.

Fit the 3D printed washer onto the on/off button.
Secure On/Off Button

Insert the On/Off button into the corresponding mounting holes onto the top half of the guitar body.

Use the hex nut to secure the On/Off button.

Checkpoint

Take a moment to ensure the various components are tightly secured in the top half of the guitar body.
Secure Feather Mount
Use 2x M3 x 10mm long screws and hex nuts to secure the Feather mount.

Place the Feather mount onto bottom half of the guitar body with the mounting lined up.

Insert the M3 screws through the guitar body and Feather mount.

Fasten the M3 hex nuts to secure the Feather mount.

Secured Feather Mount
Check the Feather mount has been properly secured to the bottom half of the guitar body.
Secure TRRS Breakout

Place the TRRS breakout over the corresponding mounting holes on the bottom half of the guitar body.

Insert and fasten 2x M2 x 6mm long screws to secure the TRRS breakout.
Cable for Rotary Encoders
Use the 3-pin picoBlade cable and solder each wire to the interrupt pins (INT) on the three rotary encoder breakouts.

- A1 Encoder to Yellow Wire
- A0 Encoder to Black Wire
- 3rd Encoder to Red Wire

Rotary Encoder Cable for Feather
Solder the wires from the matching 3-pin picoBlade cable to the D5, D6 and D9 pins on the Feather respectively.

- Red wire to D5
- Black wire to D6
- Yellow Wire to D9
Connect 3-pin PicoBlade cables together

Solder SPST Switch Signal Wire to Feather
Grab the signal wire from the SPST switch and solder it to pin D12 on the Feather.

Connect STEMMA QT to Feather
Grab the long STEMMA QT cable from the rotary encoder and plug it into the STEMMA QT port on the Feather.
Solder Micro Switch Ground Wire
Grab the ground wire from the two micro switches and solder it to one of the available ground pins on the rotary encoder breakout.

Connect cables from On/Off Button to Feather
Connect Strum Switches To Feather
Grab the signal wire from the micro switches and insert it into the corresponding screw block terminal on the Feather.

Use a flat head screw driver to secure the wire.

Connect NeoPixel to Feather
Grab the cable from the NeoPixel stick and insert them into the corresponding terminals on the Feather.

Use a flat head screw driver to secure the wires.

Connect TRRS to Feather
Grab the remaining wires from the TRRS breakout and insert them into the corresponding terminals on the Feather.

Use a flat head screw driver to secure the wires.
Install Battery Cover
Fit the battery cover into the corresponding opening on the bottom half of the guitar with the mounting tabs lined up with the mounting holes.

Use 2x M3 x 6mm long screws to secure the battery cover to the body of the guitar.

Connect Battery to Feather
Plug in the 2-pin JST PH from the battery to the Feather.
Closing Body Halves
Carefully begin to close the two halves of the guitar.

Move and adjust the various cables so they aren't being pinched or kinked.

Ensure the wires are clear from the strum mechanism.

Test the strum pick to make sure it can be actuated.

Secure Body Halves
Use 8x M3 x 10mm long screws to secure the two halves of the guitar together.

Insert M3 screws through the countersunk standoffs on the bottom half of the guitar.

Fasten screws until they're tight.
Kailh Key Switches
Get the eight kailh key switches and neck cover.

Install Neck Cover
Orient the neck cover with the neck case and firmly press together to snap fit them closed.

Install Key Switches
Orient key switches so pins are matching the sockets on the NeoKey PCBs.

Carefully press fit each key switch into the neck cover.

Repeat installation process for each key switch.
Install Key Caps
Press fit the key caps onto the stems of the kailh key switches.

Final Build
Congratulations on your Synth Guitar build! Use the on/off button to it on power.

Use the strum mode toggle to switch between modes.

Use the audio mode toggle to switch between the speaker and TRRS jack.

Use the middle rotary encoder to change the rate of the LFO.

Use the 1st rotary encoder to change the octave of the notes.

Use the 3rd rotary encoder to adjust the volume.
Use

Volume

Encoder 0 controls the volume of the synth. If you press the encoder switch it will mute the synth.

LFO Rate (Tremolo)

Encoder 1 controls the LFO rate. If you press the encoder button it enables or disables the LFO.

Note Octave

Encoder 2 controls the note frequencies. Turning the encoder increases or decreases the octave range of the guitar. Pressing the encoder button switches between triad or diatonic mode.

Strum Mode

If strum mode is enabled, it plays the currently pressed synth note. If strum mode is not enabled, then pressing the NeoKey switches will play the synth.
Audio Mode

If audio mode is enabled, the output will switch from the on-board speaker to the TRRS jack.

The RP2040 PropMaker's built in speaker driver is not designed for use with headphones or line in.

Filters

The accelerometer controls whether a high pass or low pass filter is applied to the synth. If the guitar is tilted up, a high pass filter is enabled. If the guitar is tilted down, a low pass filter is enabled. The filter frequency and resonance are affected by the readings from the accelerometer.