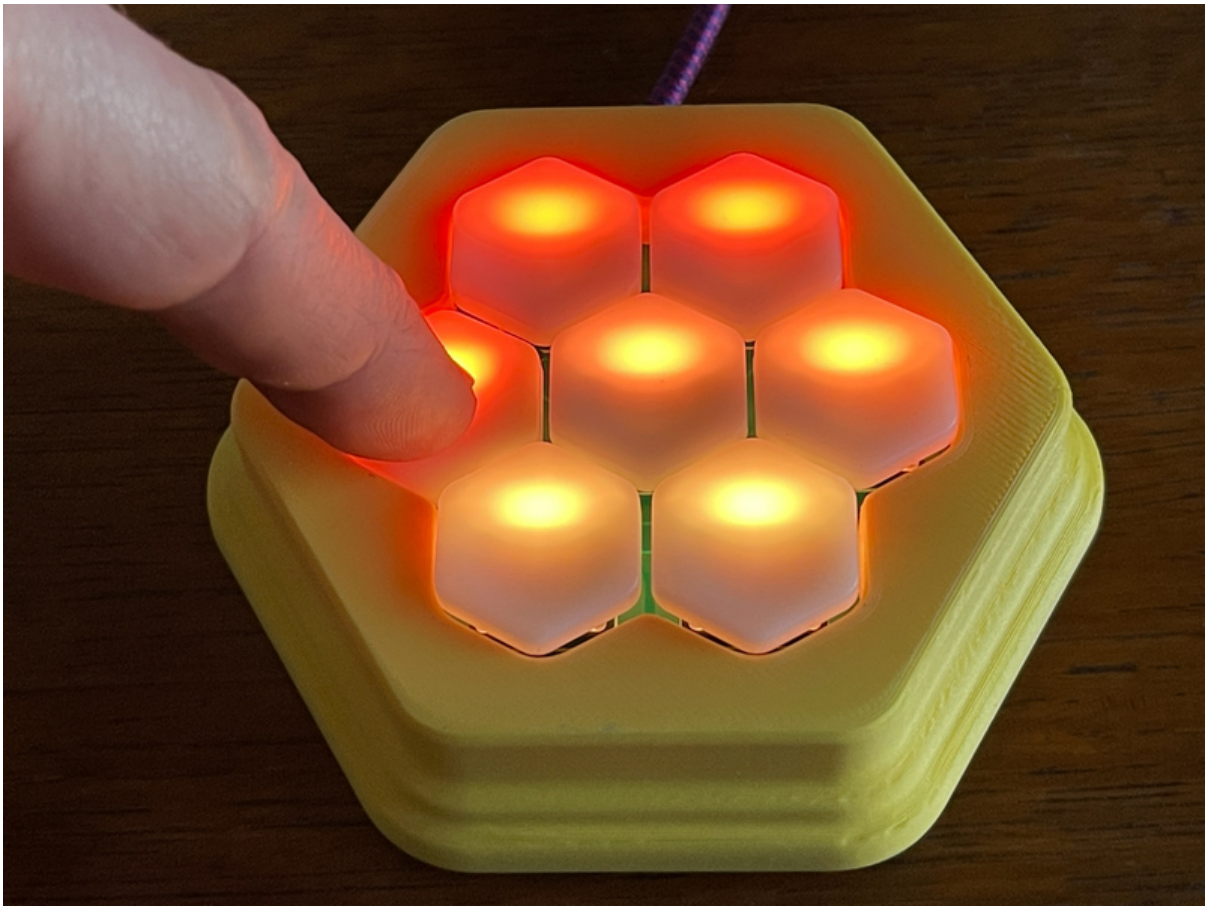
Hexpad

Created by John Park



https://learn.adafruit.com/hexpad

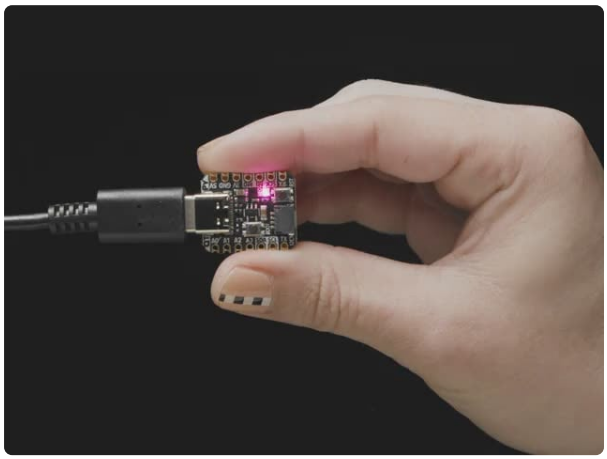
Last updated on 2024-03-08 04:12:43 PM EST

Table of Contents

Overview

Play only the "good notes" with this MIDI Hexpad. You can build a hextacular isomorphic controller using an Adafruit QT Py RP2040, low-profile Kailh CHOC keyswitches, a custom PCB, and hexagonal keyswitches, in a 3D printed case, all running on CircuitPython.

Parts

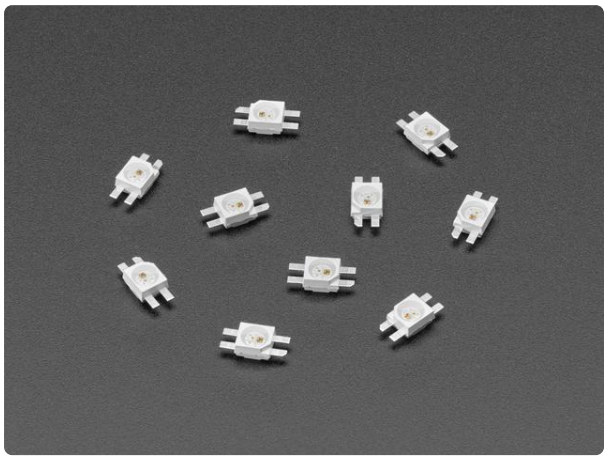


Adafruit QT Py RP2040

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

Kailh CHOC Low Profile White Clicky Key Switches

For crafting your very own custom keyboard, these Kailh Choc Low Profile Clicky White mechanical key switches are super slim, with ultra-low profile... https://www.adafruit.com/product/5114



NeoPixel Reverse Mount RGB LEDs - 10 Pack of SK6812-E

These reverse mount NeoPixel LEDs are an easy way to add a lot of small (but bright!) colorful LEDs to your project when you want the top of the PCB to be flat,... https://www.adafruit.com/product/4960

Hexagonal Choc Keycaps

These beautiful low-profile keycaps were designed by Sol Bekic, a.k.a. s-ol) and produced by FKCaps, more info available here (https://adafru.it/18AM). I ordered mine from Little Keyboards (https://adafru.it/18AN).

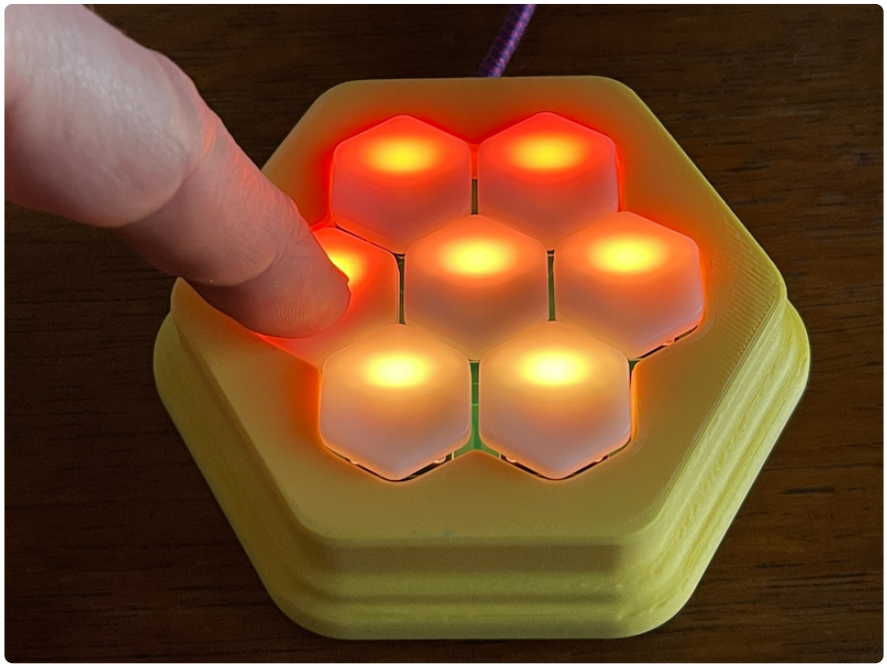
Here's an excellent build log (https://adafru.it/18AO) on the design and creation of the keycaps.

Sol's 0x33 MIDI board (https://adafru.it/18AR) was the inspiration for this build.

You can also choose to 3D print similar hex keycaps, which work particularly well with resin printing methods. More info on that, and 3D model files are available later in this guide.

Screws

6ea. of **M3 x 5mm** screws



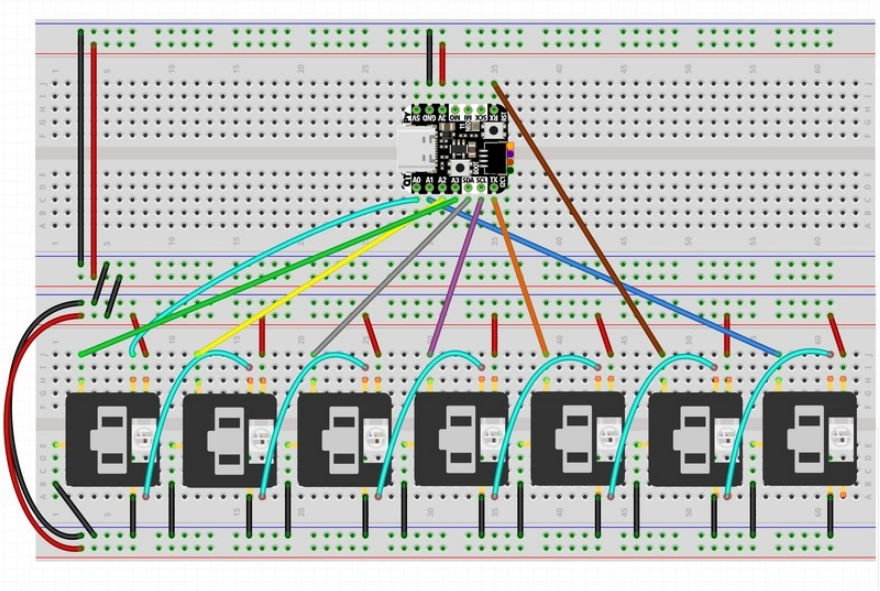
Hexpad PCB

Keyswitches aren't breadboard/protoboard friendly, so we'll create a custom PCB (printed circuit board) to build the hexpad. This will also allow us to place reverse-mount NeoPixel LEDs under each key for glow-through action.

I used Fritzing to create the parts and PCB, although you could certainly do this in your favorite board CAD software, such as Eagle, KiCad, and others.

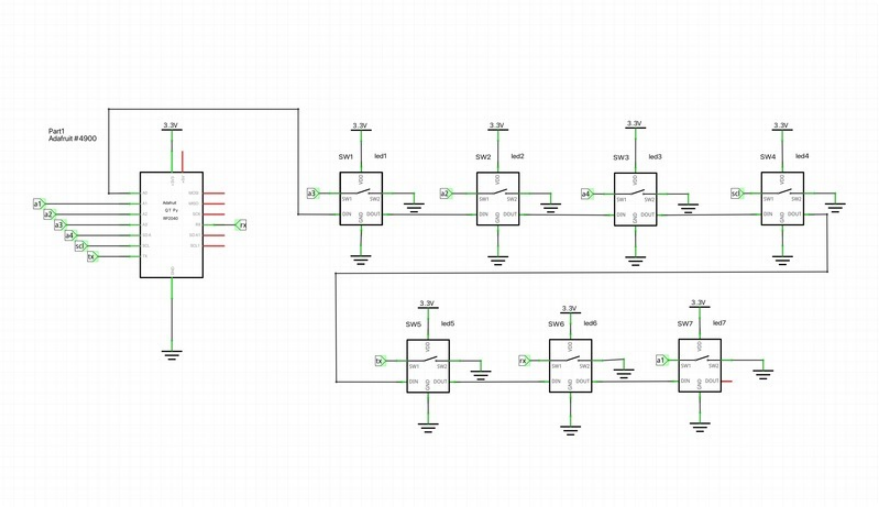
Visualization

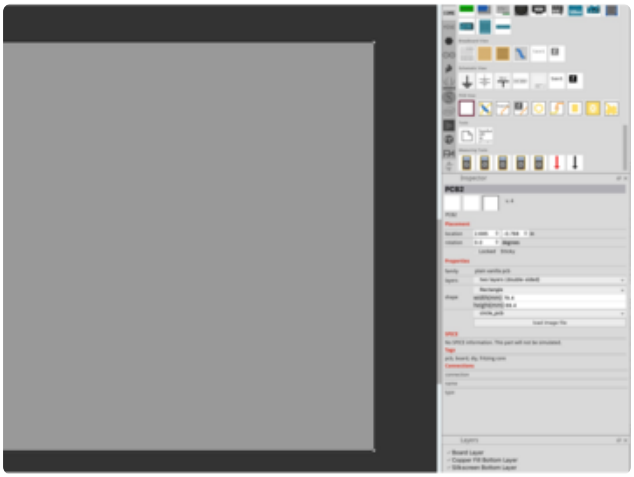
It was helpful for me to visualize the parts on a breadboard (despite the fact that you can't really use the keyswitches on a breadboard). Here you can see we have seven GPIO pin connections for the switches, and one connection for the NeoPixel data, along with power and ground.

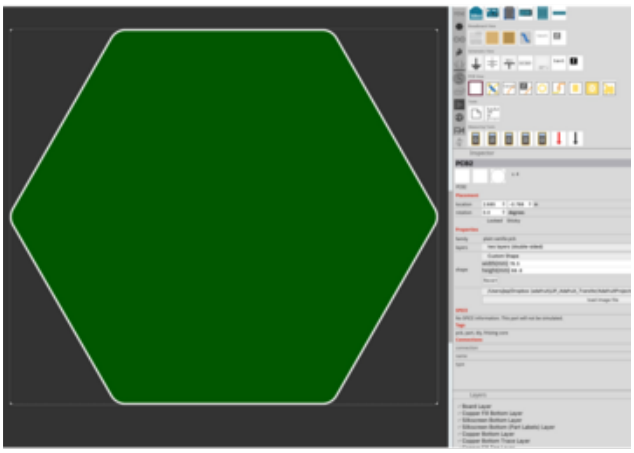


Schematic

Before creating the PCB layout it's a good idea to clearly lay out the schematic view. Note the use of net labels to keep things from getting too crowded with wires.







Custom PCB Shape

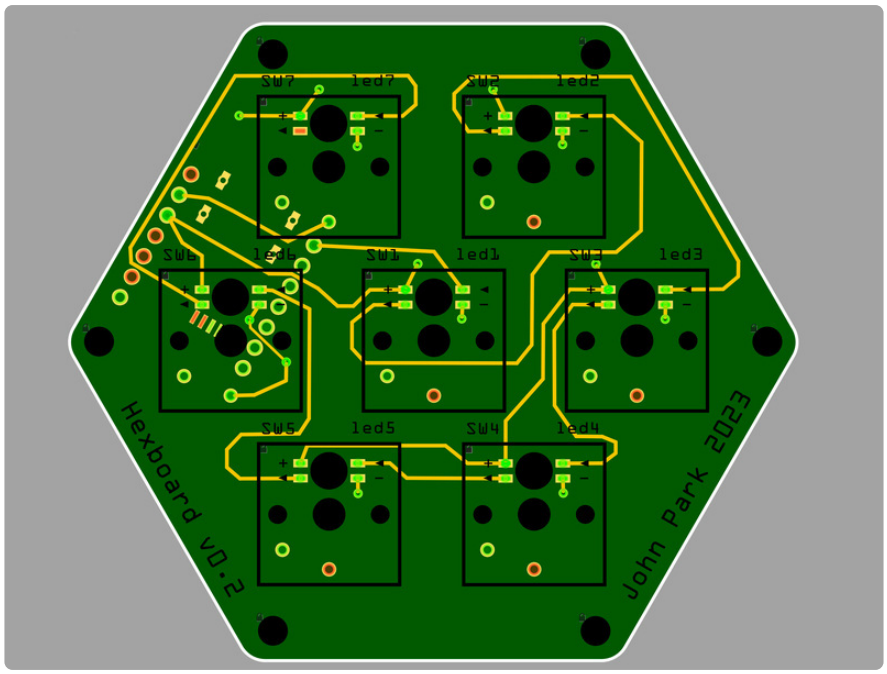
You can use a custom PCB shape in Fritzing by importing an .svg file into the board image parameter.

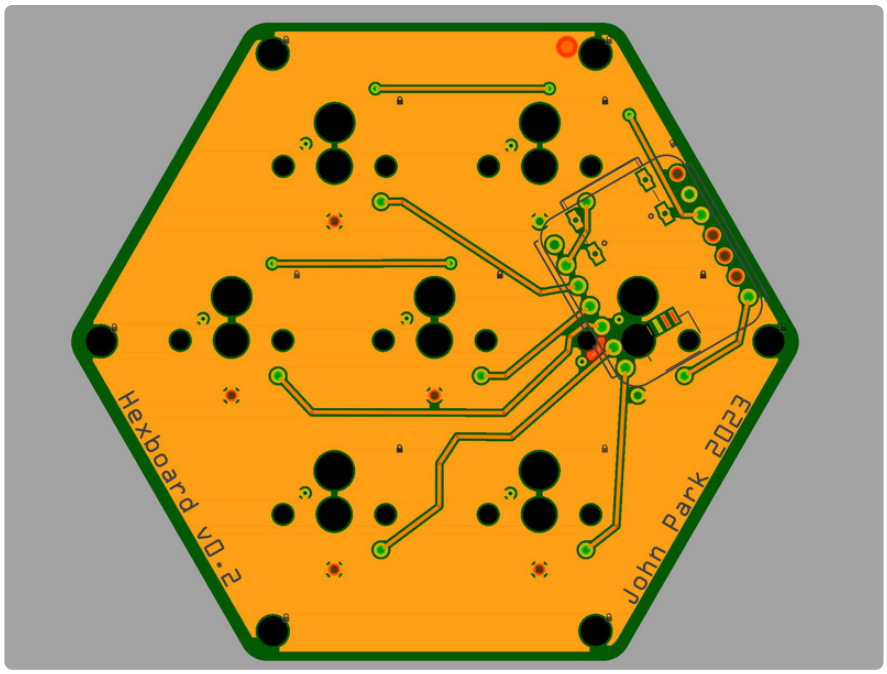
Here's a guide (https://adafru.it/18AS) with many details on creating the .svg necessary.

PCB Design in Fritzing

This guide is an introduction (https://adafru.it/QUa) to designing your PCB in Fritzing. The same techniques are used for the Hexboard.

Below you'll find the finished board and design files for download.





Hexboardv02.3.fzz

https://adafru.it/18AU

Order PCBs

This page goes through the details (https://adafru.it/18AV) of verifying your design and exporting your Gerber files. If you want to use the pre-made files, download the .zip linked below.

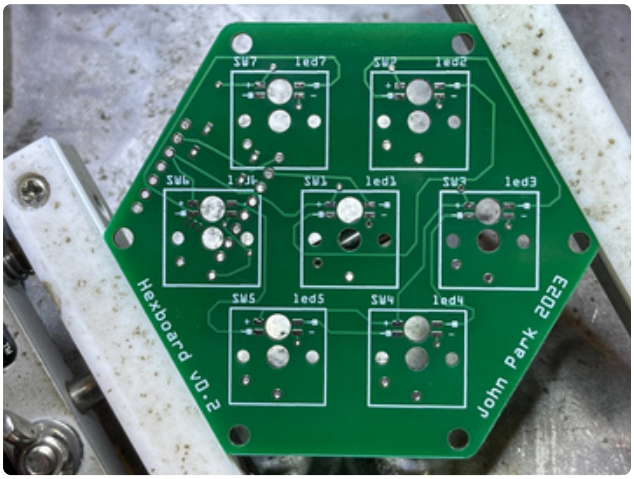
Hexboardv02_1_gerbers.zip

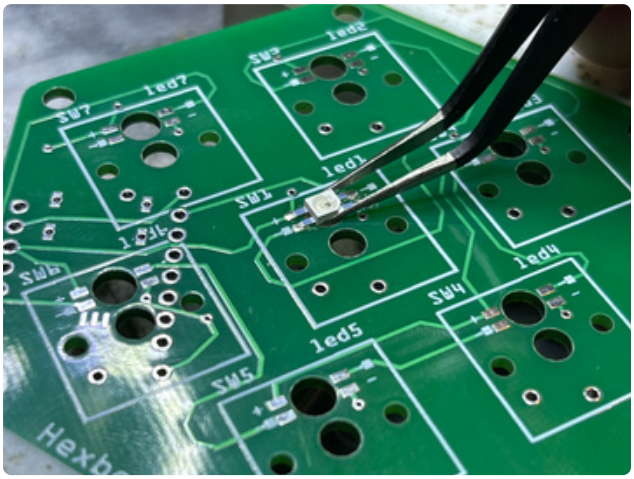
https://adafru.it/18AX

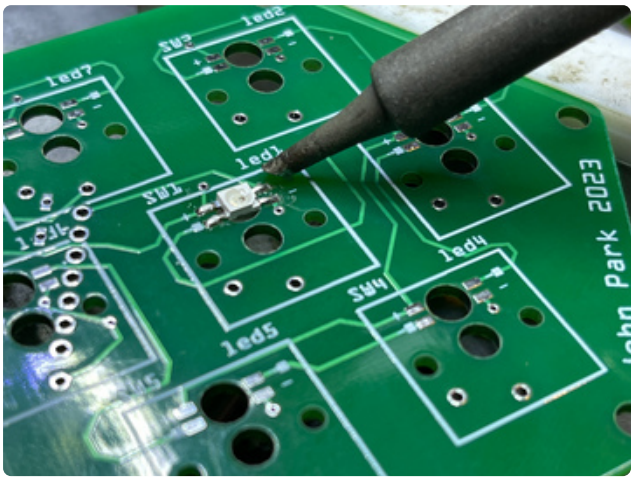
There are lots of places to have your PCBs made -- I'm a fan of both OSHPark and JLCPCB in particular, and I know people who like PCBWay a lot too. I'd recommend OSHPark for your first boards as they have terrific customer service and a great UI for helping you through the process.

Head to oshpark.com (https://adafru.it/e2G) and then drag your **Hexboardv02_1_gerber.zip** file onto the "Let's get started!" box. They'll ingest the zip, extract the files, and invite you to inspect the layers.

Build the Hexpad

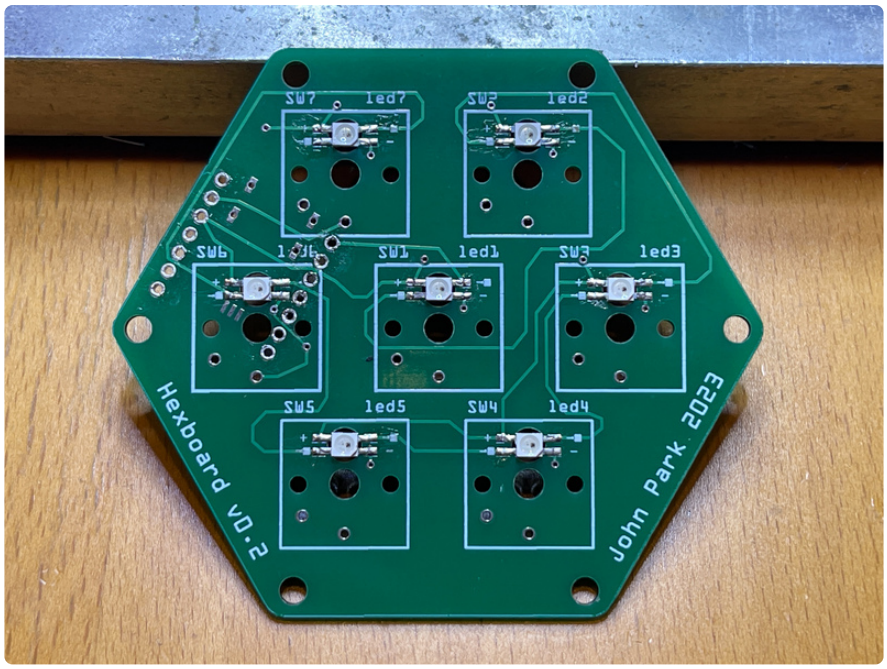


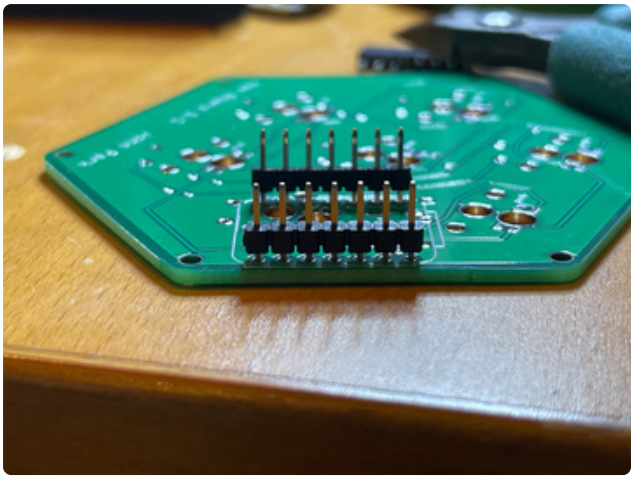


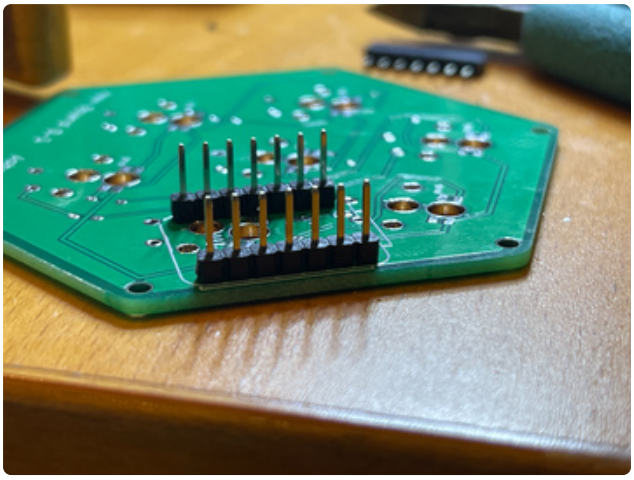


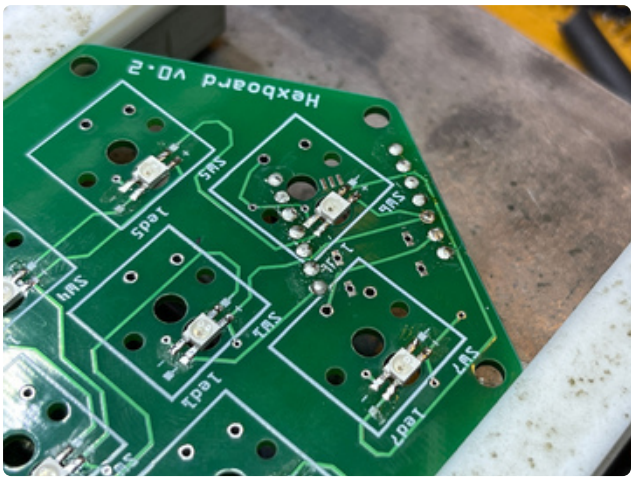
Solder the LEDs

Solder the seven NeoPixel LEDs in place. Note the location of the **"-"** pad on the silkscreen, that's where the **GND** leg on the NeoPixel that has the notched corner goes.


SW7
led7
SW2
led2
SW6
led6
SW1
led1
SW3
led3
SW5
led5
SW4
led4
Hexboard v0.2
John Park 2023







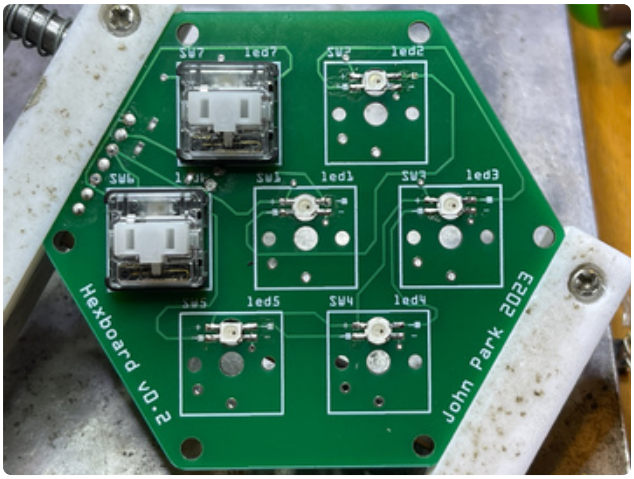
Solder the Headers

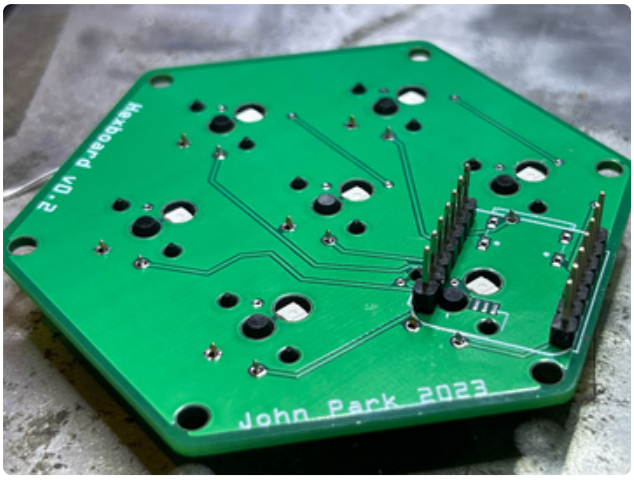
Next, you'll solder the header pins for the QT Py in place. DO NOT SOLDER THE QT PY YET!

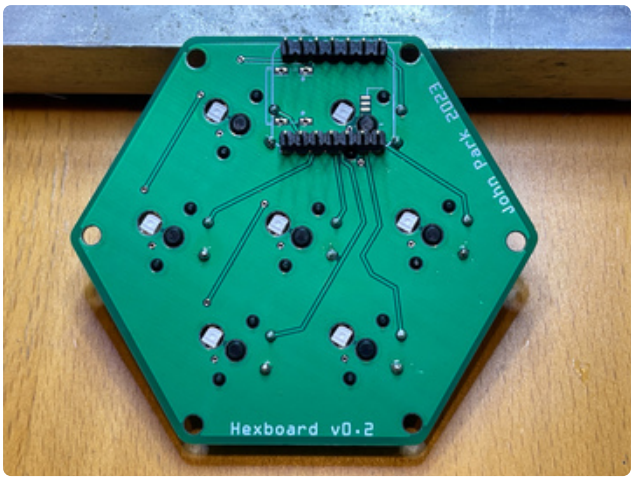
Flip the board over so the bottom side that has the QT Py outline on it is facing up.

Place two ten pin rows of headers into the holes and press the plastic spacers down so they are flush with the board. This will allow the overlapping keyswitch to be mounted flat.

Flip the board over while holding the pins in place (blue tac or tape can be helpful here) and solder them in place.



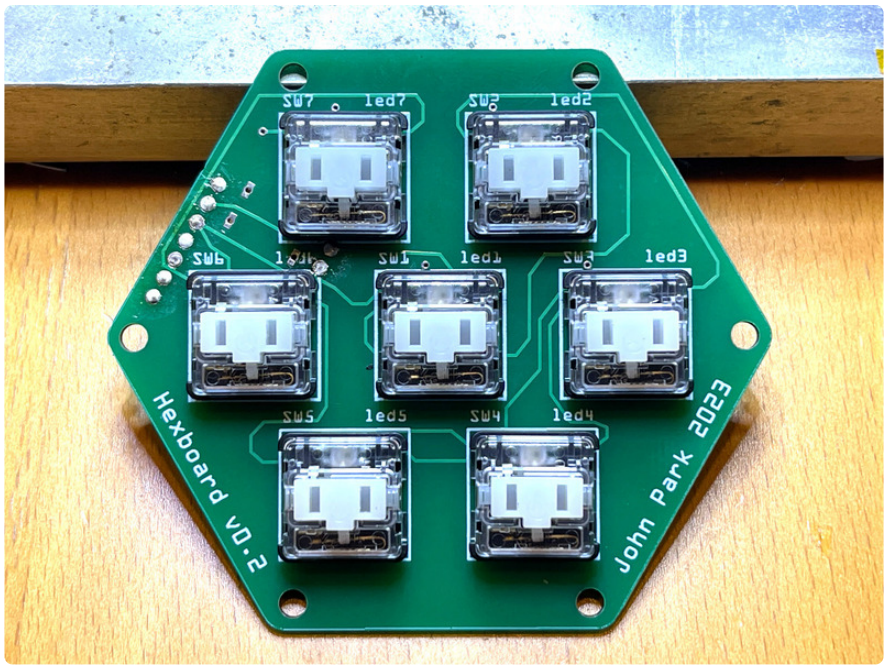


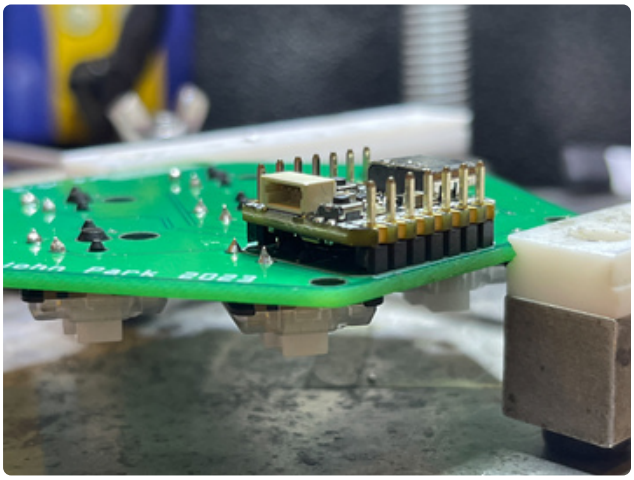


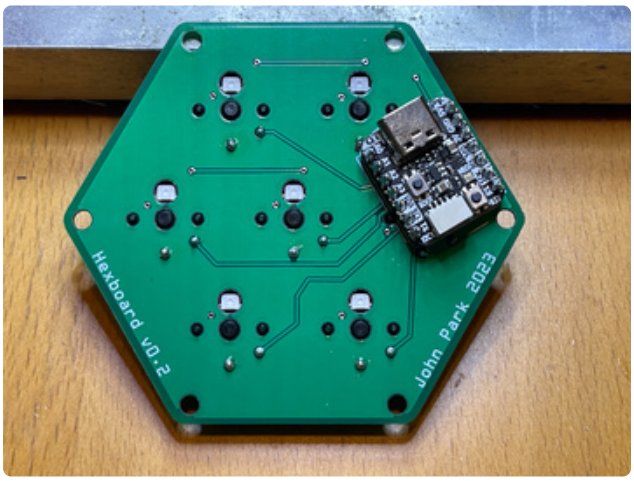
Solder the Keyswitches

Fit the keyswitches into their holes on the top side of the board (with the silkscreen outlines for the keys), being careful not to bend the fragile legs.

Flip the board over and solder the keyswitches.







Solder the QT Py

Now that the switches are soldered, you can solder the QT Py in place. Once soldered, you may optionally clip the excess header pin length.

Build the Case

3D Printing

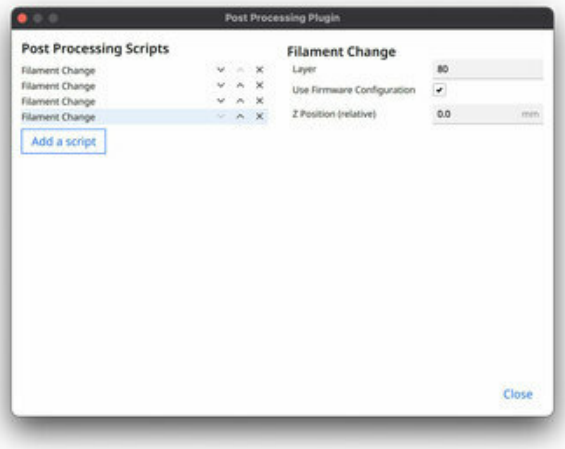
Download the case STL file using the link below. Bored of printing with just a single color of filament? Try this!

You can achieve a multi-color part with an FDM 3D printer using a "change filament" technique.

Enclosure design by John Park (https://adafru.it/18AZ)

Download Hexpad Case CAD Files

https://adafru.it/18B1



Post Processing Filament Change

In the Ultimaker CURA software, access the **Post Processing Plugin** window by going to the top menu:

Extensions > **Post Processing** > **Modify G-Code**

Click the **Add a Script** button and choose **Filament Change** from the dropdown menu.

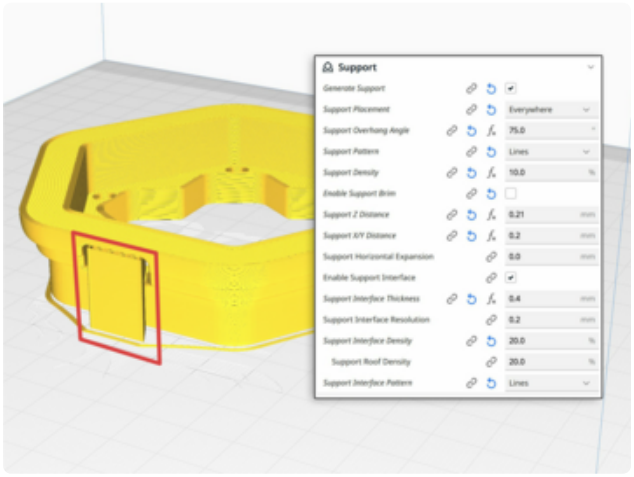
To achieve the five layers of colors, add four Filament Change scripts. Enter a value in the layer section and enable the **Use Firmware Configuration** option.

Filament Change Layers

Use a yellow colored filament for the layers 1-19, then swap out the filament for black when the 3D printer parks and goes through the change filament process.

1. Layer 20 – Black

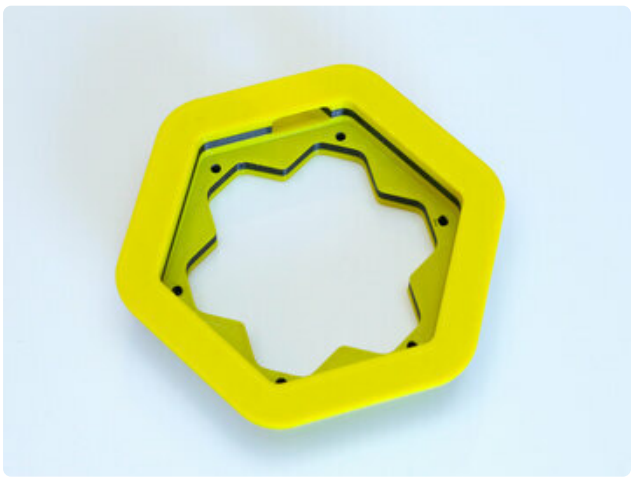
2. Layer 40 – Yellow
3. Layer 60 – Black
4. Layer 80 – Yellow



Support Material

For best print quality, use support material in the USB port area. This will help with bridging and creates a nice quality surface. Use the following settings in your slicer program.

Support Placement: Everywhere
Enable Support Interface
Support Interface Resolution: 0.2mm
Support Interface Density: 20%
Support Density: 10%

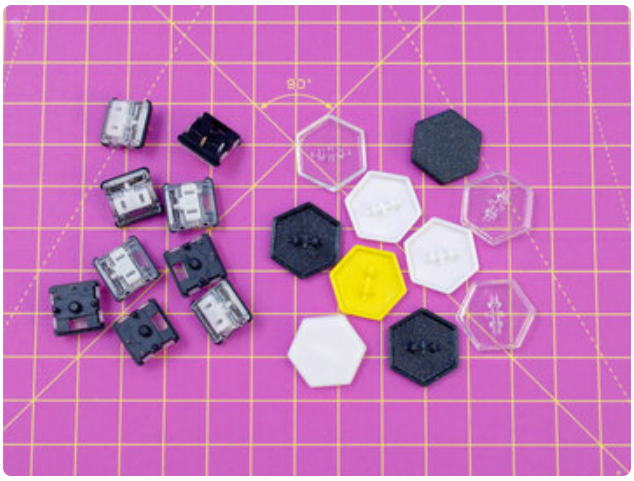


3D Printed Case

The case features an opening for a USB-C type cable for powering the QT Py RP2040.

There are six M3 sized holes for securing the hex board PCB.

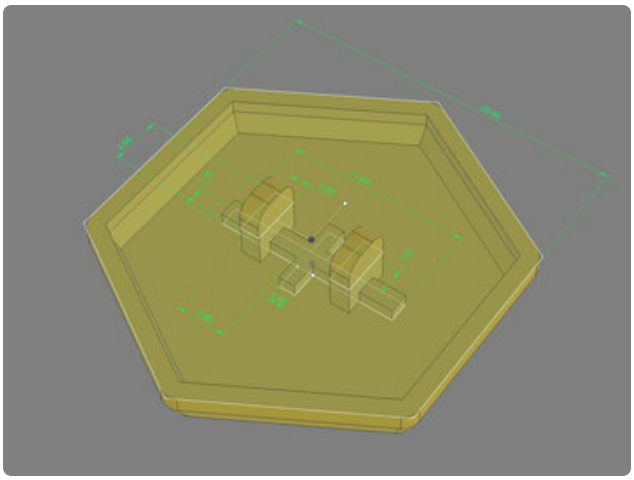
Hex Keycaps



Hexagon Keycaps

These beautiful low-profile keycaps were designed by Sol Bekic, a.k.a. s-ol and produced by FKCaps, more info available here (https://adafru.it/18AM). I ordered mine from Little Keyboards (https://adafru.it/18AN). If they're out of stock, you can 3D print your own.

The hexagon keycaps are designed to fit low profile CHOC mechanical key switches.



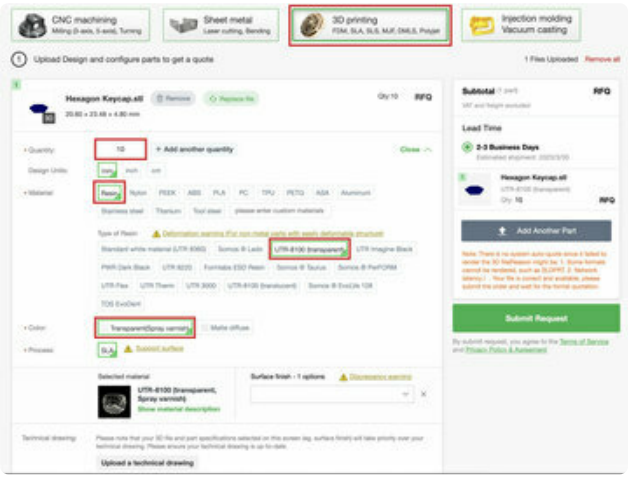
CAD Files

The hex keycaps can be 3D printed using FDM or SLA 3D printers. Use the link below to download the part model in various 3D file formats.

Hexagon Keycaps designed by Noe Ruiz (https://adafru.it/18B4)

Download Hexagon_Keycap.zip

https://adafru.it/18B6

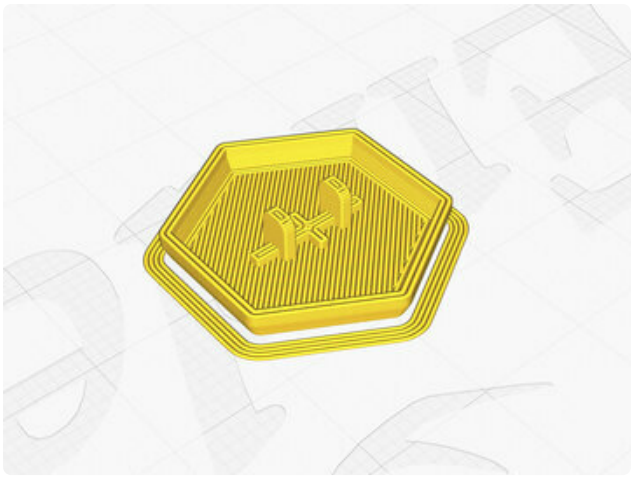


3D Printing Service

PCBWay.com (https://adafru.it/18B7) is a service that can 3D print high-quality parts in various materials and colors.

UTR-8100 transparent resin is a great option for making crystal clear key caps. Use the following settings to have PCBWay produce and ship them to you.

Enter your quantity
Material: Resin, UTR-8100 (transparent)
Color: Transparent (Spray Varnish)
Process: SLA



Slicing Keycaps

Use your preferred slicing software for 3D printing the keycaps using an FDM 3D printer.

For best illumination we suggest using a light or semi-translucent filament for the NeoPixels to shine through the material.

CircuitPython

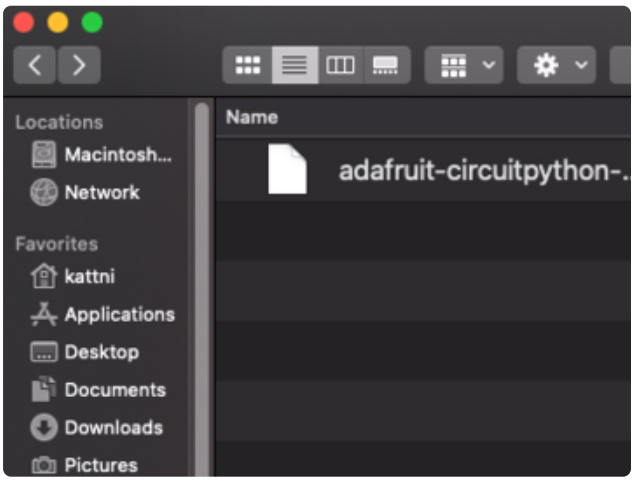
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

CircuitPython Quickstart

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

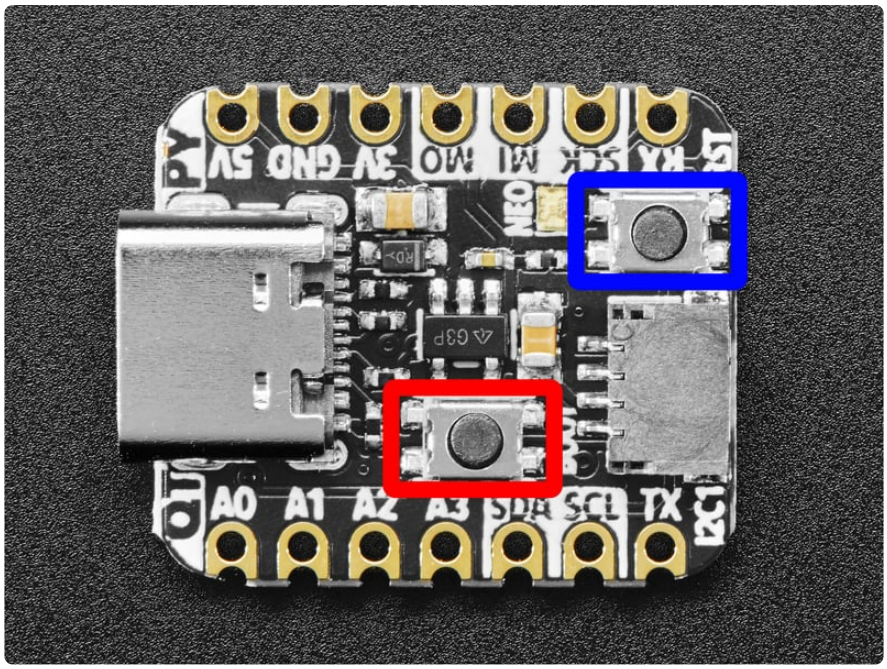
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/RLD



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

Save it wherever is convenient for you.

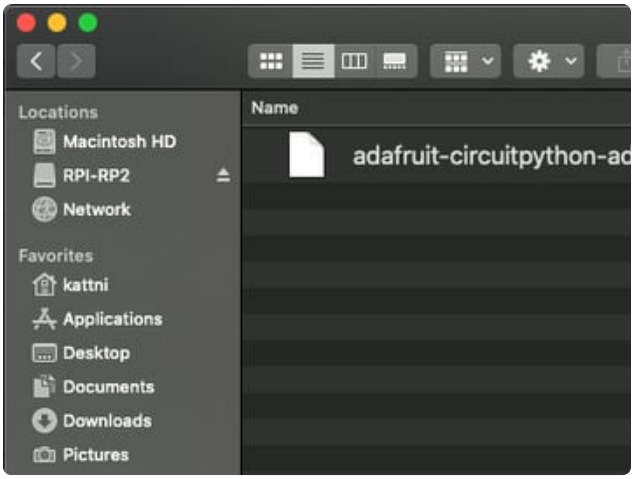


To enter the bootloader, hold down the **BOOT/BOOTSEL button** (highlighted in red above), and while continuing to hold it (don't let go!), press and release the **reset button** (highlighted in blue above). **Continue to hold the BOOT/BOOTSEL button until the RPI-RP2 drive appears!**

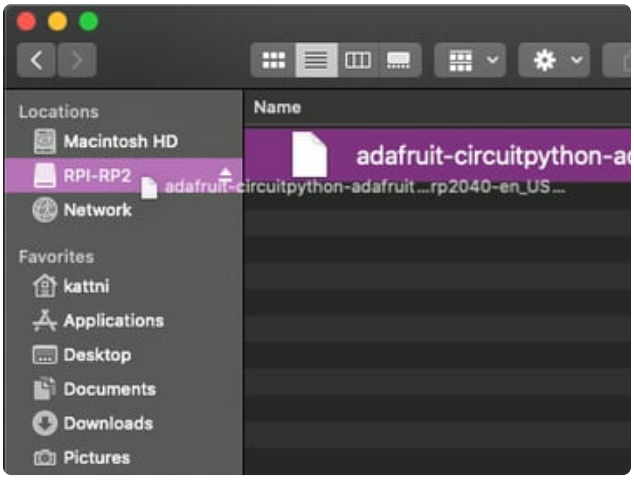
If the drive does not appear, release all the buttons, and then repeat the process above.

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.

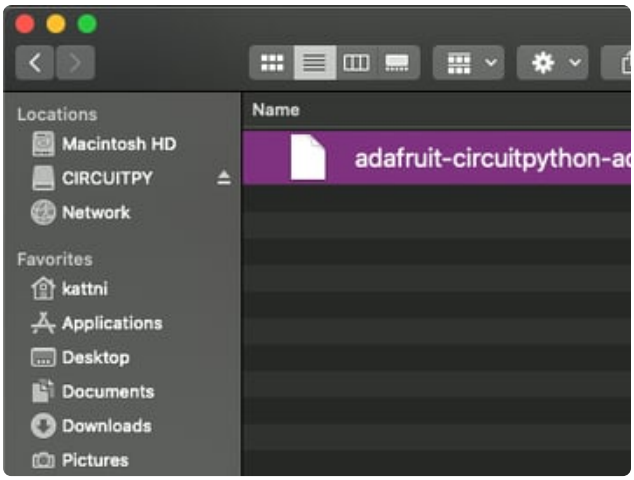
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



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

https://adafru.it/RLE

Code and Use the Hexpad

Here's a great demo by Astrophage:

Text Editor

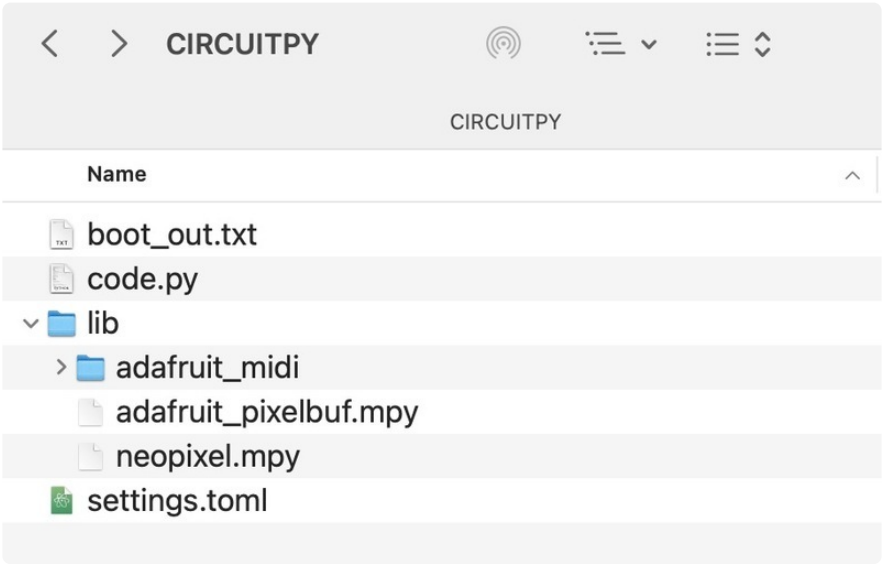
Adafruit recommends using the **Mu** editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

Alternatively, you can use any text editor that saves simple text files.

Download the Project Bundle

Your project will use a specific set of CircuitPython libraries, and the **code.py** file. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Drag the contents of the uncompressed bundle directory onto your board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.



```
# SPDX-FileCopyrightText: 2023 John Park for Adafruit
#
# SPDX-License-Identifier: MIT
# Hexboard seven key modal note/chord pad for MIDI instruments
# Runs on QT Py RP2040
# (other QT Pys should work, but the BOOT button is handy for initiating
configuration)

import time
import board
from digitalio import DigitalInOut, Pull
import keypad
import neopixel
import rainbowio
import usb_midi
import adafruit_midi
from adafruit_midi.note_on import NoteOn
from adafruit_midi.note_off import NoteOff

button = DigitalInOut(board.BUTTON)
button.pull = Pull.UP

num_switches = 7
leds = neopixel.NeoPixel(board.A0, num_switches, brightness=0.7)
leds.fill(rainbowio.colorwheel(5))
leds.show()

# root_picked = False
note = 0
root = 0  # defaults to a C

#  lists of modal intervals (relative to root). Customize these if you want other
scales/keys
major = (0, 2, 4, 5, 7, 9, 11)
minor = (0, 2, 3, 5, 7, 8, 10)
dorian = (0, 2, 3, 5, 7, 9, 10)
phrygian = (0, 1, 3, 5, 7, 8, 10)
lydian = (0, 2, 4, 6, 7, 9, 11)
mixolydian = (0, 2, 4, 5, 7, 9, 10)
locrian = (0, 1, 3, 5, 6, 8, 10)

modes = []
modes.append(major)
modes.append(minor)
modes.append(dorian)
modes.append(phrygian)
modes.append(lydian)
modes.append(mixolydian)
modes.append(locrian)

octv = 4
mode = 0  # default to major scale
play_chords = True  # default to play chords
pre_notes = modes[mode]  # initial mapping
keymap = (4, 3, 5, 0, 2, 6, 1)  # physical to logical key mapping

#  Key chart  | logical  |Interval chart example
#    6   1    |  6   7   |   9  11
#   5  0  2   | 3  4  5  |  4   5   7
#    4   3    |  0   1   |    0   2

# MIDI Setup
midi_usb_channel = 1  # change this to your desired MIDI out channel, 1-16
midi_usb = adafruit_midi.MIDI(midi_out=usb_midi.ports[1],
out_channel=midi_usb_channel-1)

# Keyswitch setup
keyswitch_pins = (board.A3, board.A2, board.SDA, board.SCL, board.TX, board.RX,
```

```
board.A1)
keyswitches = keypad.Keys(keyswitch_pins, value_when_pressed=False, pull=True)

def pick_mode():
    print("Choose mode...")
    mode_picked = False
    # pylint: disable=global-statement
    global mode
    while not mode_picked:
        # pylint: disable=redefined-outer-name
        keyswitch = keyswitches.events.get()  # check for key events
        if keyswitch:
            if keyswitch.pressed:
                mode = keymap.index(keyswitch.key_number)  # bottom left key is 0/
major
                print("Mode is:", mode)
            if keyswitch.released:
                mode_picked = True
                leds.fill(rainbowio.colorwheel(8))
                leds.show()
                pick_octave()

def pick_octave():
    print("Choose octave...")
    octave_picked = False
    # pylint: disable=global-statement
    global octv
    while not octave_picked:
        if button.value is False:  # pressed
            launch_config()
            time.sleep(0.1)
        # pylint: disable=redefined-outer-name
        keyswitch = keyswitches.events.get()  # check for key events
        if keyswitch:
            if keyswitch.pressed:
                octv = keymap.index(keyswitch.key_number)  # get remapped position,
lower left is 0
                print("Octave is:", octv)
            if keyswitch.released:
                octave_picked = True
                leds.fill(rainbowio.colorwheel(16))
                pick_root()

def pick_root():# user selects key in which to play
    print("Choose root note...")
    root_picked = False
    # pylint: disable=global-statement
    global root
    while not root_picked:
        if button.value is False:  # pressed
            launch_config()
            time.sleep(0.1)
        # pylint: disable=redefined-outer-name
        keyswitch = keyswitches.events.get()  # check for key events
        if keyswitch:
            if keyswitch.pressed:
                root = keymap.index(keyswitch.key_number)  # get remapped position,
lower left is 0
                print("ksw:", keyswitch.key_number, "keymap index:", root)
                note = pre_notes[root]
                print("note:", note)
                midi_usb.send(NoteOn(note + (12*octv), 120))
                root_notes.clear()
                # pylint: disable=redefined-outer-name
                for mode_interval in range(num_switches):
                    root_notes.append(modes[mode][mode_interval] + note)
                print("root note intervals:", root_notes)
            if keyswitch.released:
                note = pre_notes[root]
```

```
                midi_usb.send(NoteOff(note + (12*octv), 0))
                root_picked = True
                leds.fill(0x0)
                leds[3] = rainbowio.colorwheel(12)
                leds[4] = rainbowio.colorwheel(5)
                leds.show()
                pick_chords()

def pick_chords():
    print("Choose chords vs. single notes...")
    chords_picked = False
    # pylint: disable=global-statement
    global play_chords
    while not chords_picked:
        if button.value is False:  # pressed
            launch_config()
            time.sleep(0.1)
        # pylint: disable=redefined-outer-name
        keyswitch = keyswitches.events.get()  # check for key events
        if keyswitch:
            if keyswitch.pressed:
                if keyswitch.key_number == 4:
                    play_chords = True
                    print("Chords are on")
                    chords_picked = True
                    playback_led_colors()
                if keyswitch.key_number == 3:
                    play_chords = False
                    print("Chords are off")
                    chords_picked = True
                    playback_led_colors()

# create the interval list based on root key and mode that's been picked in variable
root_notes = []
for mode_interval in range(num_switches):
    root_notes.append(modes[mode][mode_interval] + note)
print("---Hexpad---")
print("\nRoot note intervals:", root_notes)

key_colors = (18, 10, 18, 26, 26, 18, 10)

def playback_led_colors():
    for i in range(num_switches):
        leds[i]=(rainbowio.colorwheel(key_colors[i]))
        leds.show()
        time.sleep(0.1)

playback_led_colors()

# MIDI Note Message Functions
def send_note_on(note_num):
    if play_chords is True:
        note_num = root_notes[note_num] + (12*octv)
        midi_usb.send(NoteOn(note_num, 120))
        midi_usb.send(NoteOn(note_num + modes[mode][2], 80))
        midi_usb.send(NoteOn(note_num + modes[mode][4], 60))
        midi_usb.send(NoteOn(note_num+12, 80))
    else:
        note_num = root_notes[note_num] + (12*octv)
        midi_usb.send(NoteOn(note_num, 120))


def send_note_off(note_num):
    if play_chords is True:
        note_num = root_notes[note_num] + (12*octv)
        midi_usb.send(NoteOff(note_num, 0))
        midi_usb.send(NoteOff(note_num + modes[mode][2], 0))
        midi_usb.send(NoteOff(note_num + modes[mode][4], 0))
        midi_usb.send(NoteOff(note_num+12, 0))
```

```
    else:
        note_num = root_notes[note_num] + (12*octv)
        midi_usb.send(NoteOff(note_num, 0))

def send_midi_panic():
    for x in range(128):
        midi_usb.send(NoteOff(x, 0))

def launch_config():
    print("-launching config-")
    send_midi_panic()
    leds.fill(rainbowio.colorwheel(5))
    leds.show()
    pick_mode()

send_midi_panic()  # turn off any stuck notes at startup


while True:
    keyswitch = keyswitches.events.get()  # check for key events
    if keyswitch:
        keyswitch_number=keyswitch.key_number
        if keyswitch.pressed:
            note_picked = keymap.index(keyswitch.key_number)
            send_note_on(note_picked)
            leds[keyswitch_number]=(rainbowio.colorwheel(10))

            leds.show()
        if keyswitch.released:
            note_picked = keymap.index(keyswitch.key_number)
            send_note_off(note_picked)

leds[keyswitch_number]=(rainbowio.colorwheel(key_colors[keyswitch_number]))
            leds.show()

    if button.value is False:  # pressed
        launch_config()
        time.sleep(0.1)
```

Play the Hexpad

You may already have a favorite software synth, and chances are it'll work with the Hexpad. In case you don't have one already picked out, here are some good ones to try.

iOS (with an OTG USB to Lightning adapter (http://adafru.it/3940))

- AudioKit Synth One (https://adafru.it/C-6)

Chrome Web Browser

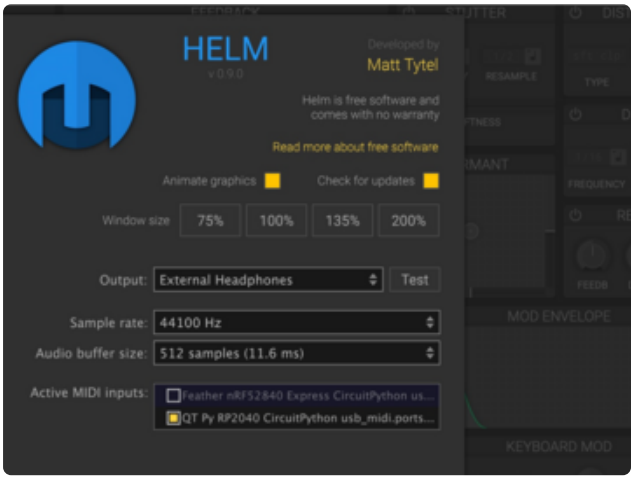
- Viktor NV-1
- Juno-106.js (https://adafru.it/C-5)

Linux / Windows / mac os

- Helm (https://adafru.it/C-a)
- VCV Rack (https://adafru.it/C-b)
- Pure Data (https://adafru.it/C-c)
- Ardour (https://adafru.it/C-d)

Once you've picked a synth, plug in your Hexpad and get playing!

Plug in a known good power and data USB-C cable to the Hexpad, and plug the other end into your computer or OTG adapter.



MIDI Device

It's very likely the synth will recognize it immediately, but if not, check the preferences and choose the "QT Py RP2040" MIDI device.

Now, press any keys or combinations to play! All notes will be "good" notes because of the scale mode.

If you want to change the mode, octave, root note, and the choice of modal chords vs. single notes, simply press the Boot button on the QT Py. You'll see a prompt for each step in a serial REPL window, but you can also do these steps without it.

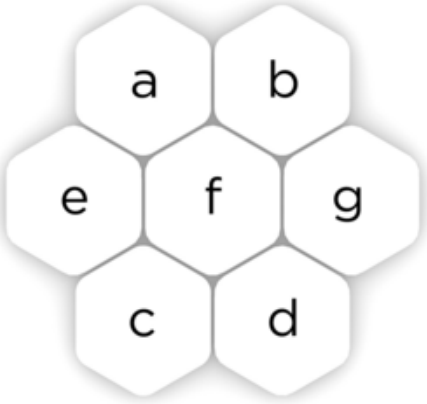
These are the configuration steps and corresponding keys:

Mode Map

key 0 = **major**
key 1 = **minor**
key 2 = **dorian**
key 3 = **phrygian**
key 4 = **lydian**
key 5 = **mixolydian**
key 6 = **locrian**
You can create other modes by editing the **code.py** file directly.

Octave Map

The octaves correspond to the key numbers shown here. 4 or 3 are good places to start, but you can play super low or ultra high if you like!



Root Map

The key-to-note assignment will vary during play depending on the root note and mode. While in configuration mode on the root selection step, these are the assignments.

You can adjust these in code if you want sharps/flats.

Chord Mode

For polyphonic chord playback, press the **chord** key. For monophonic note playback, press the **single** key.

How it Works

The code first imports libraries, including keypad, neopixel, and the MIDI libraries.

```
import time
import board
from digitalio import DigitalInOut, Pull
import keypad
import neopixel
import rainbowio
import usb_midi
import adafruit_midi
from adafruit_midi.note_on import NoteOn
from adafruit_midi.note_off import NoteOff
```

Boot Button

The **Boot** button on the QT Py RP2040 can be used as a user button. You'll set it up so that it can be used later to initiate the configuration process.

```
button = DigitalInOut(board.BUTTON)
button.pull = Pull.UP
```

LED Setup

Next the NeoPixels are set up and lit.

```
num_switches = 7
leds = neopixel.NeoPixel(board.A0, num_switches, brightness=0.7)
leds.fill(rainbowio.colorwheel(5))
leds.show()
```

Note and Mode Variables

A number of variables and lists are set up to store the note values and intervals of the scale modes.

```
note = 0
root = 0  # defaults to a C

#  lists of modal intervals (relative to root). Customize these if you want other
scales/keys
major = (0, 2, 4, 5, 7, 9, 11)
minor = (0, 2, 3, 5, 7, 8, 10)
dorian = (0, 2, 3, 5, 7, 9, 10)
phrygian = (0, 1, 3, 5, 7, 8, 10)
lydian = (0, 2, 4, 6, 7, 9, 11)
mixolydian = (0, 2, 4, 5, 7, 9, 10)
locrian = (0, 1, 3, 5, 6, 8, 10)

modes = []
modes.append(major)
modes.append(minor)
modes.append(dorian)
modes.append(phrygian)
modes.append(lydian)
modes.append(mixolydian)
modes.append(locrian)

octv = 4
mode = 0  # default to major scale
play_chords = True  # default to play chords
pre_notes = modes[mode]  # initial mapping
keymap = (4, 3, 5, 0, 2, 6, 1)  # physical to logical key mapping
```

MIDI Setup

Next, the MIDI object is set up. Here you can also change the output channel from 1-16.

```
midi_usb_channel = 1  # change this to your desired MIDI out channel, 1-16
midi_usb = adafruit_midi.MIDI(midi_out=usb_midi.ports[1],
out_channel=midi_usb_channel-1)
```

Keyswitch Setup

The keyswitches are set up using the **keypad** library, with their GPIO pins selected in order so the physical placement will correspond to the logical key assignments 0-6.

```
keyswitch_pins = (board.A3, board.A2, board.SDA, board.SCL, board.TX, board.RX,
board.A1)
keyswitches = keypad.Keys(keyswitch_pins, value_when_pressed=False, pull=True)
```

Configuration Functions

These functions are used during optional configuration (initiated by the user by pressing the **Boot** button).

Note the use of `while not mode_picked` (and others) to have the code wait for the user to press a button.

```
def pick_mode():
    print("Choose mode...")
    mode_picked = False
    # pylint: disable=global-statement
    global mode
    while not mode_picked:
        # pylint: disable=redefined-outer-name
        keyswitch = keyswitches.events.get()  # check for key events
        if keyswitch:
            if keyswitch.pressed:
                mode = keymap.index(keyswitch.key_number)  # bottom left key is 0/
major
                print("Mode is:", mode)
            if keyswitch.released:
                mode_picked = True
                leds.fill(rainbowio.colorwheel(8))
                leds.show()
                pick_octave()

def pick_octave():
    print("Choose octave...")
    octave_picked = False
    # pylint: disable=global-statement
    global octv
    while not octave_picked:
        if button.value is False:  # pressed
            launch_config()
            time.sleep(0.1)
        # pylint: disable=redefined-outer-name
        keyswitch = keyswitches.events.get()  # check for key events
        if keyswitch:
            if keyswitch.pressed:
                octv = keymap.index(keyswitch.key_number)  # get remapped position,
lower left is 0
                print("Octave is:", octv)
            if keyswitch.released:
                octave_picked = True
                leds.fill(rainbowio.colorwheel(16))
                pick_root()

def pick_root():# user selects key in which to play
    print("Choose root note...")
    root_picked = False
    # pylint: disable=global-statement
    global root
    while not root_picked:
        if button.value is False:  # pressed
            launch_config()
            time.sleep(0.1)
        # pylint: disable=redefined-outer-name
        keyswitch = keyswitches.events.get()  # check for key events
        if keyswitch:
            if keyswitch.pressed:
                root = keymap.index(keyswitch.key_number)  # get remapped position,
lower left is 0
```

```
                print("ksw:", keyswitch.key_number, "keymap index:", root)
                note = pre_notes[root]
                print("note:", note)
                midi_usb.send(NoteOn(note + (12*octv), 120))
                root_notes.clear()
                # pylint: disable=redefined-outer-name
                for mode_interval in range(num_switches):
                    root_notes.append(modes[mode][mode_interval] + note)
                print("root note intervals:", root_notes)
            if keyswitch.released:
                note = pre_notes[root]
                midi_usb.send(NoteOff(note + (12*octv), 0))
                root_picked = True
                leds.fill(0x0)
                leds[3] = rainbowio.colorwheel(12)
                leds[4] = rainbowio.colorwheel(5)
                leds.show()
                pick_chords()

def pick_chords():
    print("Choose chords vs. single notes...")
    chords_picked = False
    # pylint: disable=global-statement
    global play_chords
    while not chords_picked:
        if button.value is False:  # pressed
            launch_config()
            time.sleep(0.1)
        # pylint: disable=redefined-outer-name
        keyswitch = keyswitches.events.get()  # check for key events
        if keyswitch:
            if keyswitch.pressed:
                if keyswitch.key_number == 4:
                    play_chords = True
                    print("Chords are on")
                    chords_picked = True
                    playback_led_colors()
                if keyswitch.key_number == 3:
                    play_chords = False
                    print("Chords are off")
                    chords_picked = True
                    playback_led_colors()

def launch_config():
    print("-launching config-")
    send_midi_panic()
    leds.fill(rainbowio.colorwheel(5))
    leds.show()
    pick_mode()
```

Root Notes in Scale Mode

This list is created to put the proper notes for the chosen root and mode.

```
root_notes = []
for mode_interval in range(num_switches):
    root_notes.append(modes[mode][mode_interval] + note)
```

LED Colors

These are the color assignments per NeoPixel, all specified as values in the `rainbowio.colorwheel`.

They are all set when the Hexpad is in playback mode (at start and after configuration) with the `playback_led_colors()` function.

```
key_colors = (18, 10, 18, 26, 26, 18, 10)

def playback_led_colors():
    for i in range(num_switches):
        leds[i]=(rainbowio.colorwheel(key_colors[i]))
        leds.show()
        time.sleep(0.1)
```

Note Functions

These functions are used for playing notes/chords as well as sending MIDI panic at reset to turn off all notes.

```
def send_note_on(note_num):
    if play_chords is True:
        note_num = root_notes[note_num] + (12*octv)
        midi_usb.send(NoteOn(note_num, 120))
        midi_usb.send(NoteOn(note_num + modes[mode][2], 80))
        midi_usb.send(NoteOn(note_num + modes[mode][4], 60))
        midi_usb.send(NoteOn(note_num+12, 80))
    else:
        note_num = root_notes[note_num] + (12*octv)
        midi_usb.send(NoteOn(note_num, 120))


def send_note_off(note_num):
    if play_chords is True:
        note_num = root_notes[note_num] + (12*octv)
        midi_usb.send(NoteOff(note_num, 0))
        midi_usb.send(NoteOff(note_num + modes[mode][2], 0))
        midi_usb.send(NoteOff(note_num + modes[mode][4], 0))
        midi_usb.send(NoteOff(note_num+12, 0))
    else:
        note_num = root_notes[note_num] + (12*octv)
        midi_usb.send(NoteOff(note, 0))

def send_midi_panic():
    for x in range(128):
        midi_usb.send(NoteOff(x, 0))
```

Main Loop

The program checks for keyswitch events and plays/releases the corresponding notes. You can press multiple keys to send chords and even "chords of chords".

The main loop also checks for the **boot** button to be pressed to launch configuration.

```
while True:
    keyswitch = keyswitches.events.get()  # check for key events
    if keyswitch:
        keyswitch_number=keyswitch.key_number
        if keyswitch.pressed:
            note_picked = keymap.index(keyswitch.key_number)
            send_note_on(note_picked)
            leds[keyswitch_number]=(rainbowio.colorwheel(10))

            leds.show()
        if keyswitch.released:
            note_picked = keymap.index(keyswitch.key_number)
            send_note_off(note_picked)

leds[keyswitch_number]=(rainbowio.colorwheel(key_colors[keyswitch_number]))
            leds.show()

    if button.value is False:  # pressed
        launch_config()
        time.sleep(0.1)
```