CircuitPython MIDI to CV Skull
Created by Liz Clark

https://learn.adafruit.com/circuitpython-midi-to-cv-skull

Last updated on 2023-08-29 04:39:07 PM EDT
# Table of Contents

**Overview**  
- What is 1V/oct?  
- Prerequisite Guides  
- Parts

**Circuit Diagram**  
- Why Not Use a STEMMA QT Cable?

**3D Printing**

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

**Code the MIDI to CV Skull**  
- Upload the Code and Libraries to the QT Py RP2040  
- volts.py File  
- How the CircuitPython Code Works  
- Mapping Voltages  
- The Loop

**Wiring**  
- Components  
- Ground Rail  
- Power Rail  
- Gate Out Signal  
- The MCP4725 DAC  
- DAC Power  
- DAC Ground  
- DAC I2C  
- DAC Output

**Assembly**  
- Prepare the Skull  
- Prepare the Boards  
- Attach the Boards  
- Finish

**Usage**  
- Going Further - Winterbloom's CircuitPython Eurorack Modules
Overview

What do you do if you want the convenience of MIDI but the warm tones of analog synths? Build a MIDI to CV converter, of course! In this project, you'll use a QT Py RP2040 running CircuitPython with an MCP4725 DAC to convert MIDI messages to a gate output and 1V/oct output.

The build is housed in a snazzy 3D printed skull adorned enclosure. The output jacks are located in the skull's eyes.
The CircuitPython code converts MIDI \texttt{NoteOn} and \texttt{NoteOff} messages to gate out and 1V/oct signals that are sent through the two mono audio jacks. This lets you play your CV controlled synths with a MIDI keyboard or a MIDI song file from your DAW.

The MCP4725 DAC is a digital to analog converter and outputs the voltages to correspond with the expected 1V/oct value.

What is 1V/oct?

1V/oct is a standard for controlling pitch in modular synthesis. A volt is divided to correspond with the twelve pitches in western music. For example, 0.333V is pitch E-2 and 0.917V is pitch B-2. When the voltage is increased by 1V, the octave increases by 1. For example, 1.333V is pitch E-3 (an increase of 1V and 1 octave) and 2.917V is pitch B-4 (an increase of 2V and 2 octaves).

However, these pitches are approximate and depend on a variety of factors, such as your oscillator's tuning and any electrical interference. But that's half the fun of synthesis!

Prerequisite Guides

- Adafruit QT Py RP2040
- MCP4725 12-Bit DAC Tutorial
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

**Vertical Breadboard-Friendly 3.5mm Mono Headphone Jack**
Pipe audio in or out of your project with this very handy breadboard-friendly audio jack. It's a mono jack with a disconnect pin so you'll get a ground sleeve pin, a left tip...
https://www.adafruit.com/product/4031

**MCP4725 Breakout Board - 12-Bit DAC with I2C Interface**
Your microcontroller probably has an ADC (analog -> digital converter) but does it have a DAC (digital -> analog converter)??!? Now it can! This breakout board features the...
https://www.adafruit.com/product/935
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x Short Break-away Male Header</td>
<td>36-pin 0.1&quot; Short Break-away Male Header - Pack of 10</td>
<td><a href="https://www.adafruit.com/product/3009">https://www.adafruit.com/product/3009</a></td>
</tr>
<tr>
<td>1 x Short Female Header</td>
<td>36-pin 0.1&quot; Short Female Header - Pack of 5</td>
<td><a href="https://www.adafruit.com/product/3008">https://www.adafruit.com/product/3008</a></td>
</tr>
<tr>
<td>1 x M2.5 screws</td>
<td>M2.5 screws and stand-offs</td>
<td><a href="https://www.adafruit.com/product/3299">https://www.adafruit.com/product/3299</a></td>
</tr>
</tbody>
</table>

Anafruit Perma-Proto Quarter-sized Breadboard PCB - Single
Customers have asked us to carry basic perf-board, but we never liked the look of most basic perf: it's always crummy quality, with pads that flake off and no labeling. Then we...
https://www.adafruit.com/product/1608

Silicone Cover Stranded-Core Wire - 30AWG in Various Colors
Silicone-sheathing wire is super-flexible and soft, and its also strong! Able to handle up to 200°C and up to 600V, it will do when PVC covered wire wimps out. We like this wire...
https://www.adafruit.com/product/2051

Pink and Purple Woven USB A to USB C Cable - 1 meter long
This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also made for USB C for our modernized breakout boards, Feathers, and...
https://www.adafruit.com/product/5153
Circuit Diagram

MCP4725 DAC

- VIN to board 5V
- GND to board GND
- SCL to board SCL with 2K resistor
- SDA to board SDA with 2K resistor
- VOUT to left mono jack tip

QT Py RP2040

- GND to left mono jack sleeve
- GND to right mono jack sleeve
- A1 to right mono jack tip
The 2K resistors for SCL and SDA keep the incoming logic level at 3.3V, rather than 5V.

**Why Not Use a STEMMA QT Cable?**

The DAC is using the QT Py RP2040’s 5V output for power so that the 1V/oct signals have a range of 0 to 5V. If you do not need a range of 5V, then you can use a STEMMA QT cable to power the DAC with 3.3V and make the GND and I2C connections.

### 3D Printing

The MIDI to CV Skull may be assembled with 3D printed plates, described below. The top of the case can utilize a color swap to make the skull art pop. The pieces could also be milled or laser cut from other materials, such as wood or acrylic.

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

- [midiToCvSkullSTLs.zip](#)
- [Thingiverse download](#)
The skull art is achieved with different depths in the .STL file that can be accentuated by using a color swap at the 1mm height of the model in your slicer software.

The top and bottom plate have mounting holes for the 1/4 size perma proto and the DAC STEMMA board. Stand-offs are used to create a circuit sandwich.

---

**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 MIDI to CV Skull

Once you've finished setting up your QT Py RP2040 with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the Download Project Bundle button in the window below. It will download as a zipped folder.

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

import board
import simpleio
import adafruit_mcp4725
import usb_midi
import adafruit_midi
from digitalio import DigitalInOut, Direction
from adafruit_midi.note_off import NoteOff
from adafruit_midi.note_on import NoteOn
from volts import volts

# midi channel setup
midi_in_channel = 1
midi_out_channel = 1

# USB midi setup
midi = adafruit_midi.MIDI(
    midi_in=usb_midi.ports[0], in_channel=0, midi_out=usb_midi.ports[1],
    out_channel=0
)

# gate output pin
gate = DigitalInOut(board.A1)
gate.direction = Direction.OUTPUT

# i2c setup
i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a microcontroller
# dac setup over i2c
dac = adafruit_mcp4725.MCP4725(i2c)

dac.raw_value = 4095

# array for midi note numbers
midi_notes = []
# array for 12 bit 1v/oct values
pitches = []

# function to map 1v/oct voltages to 12 bit values
# these values are added to the pitches[] array
def map_volts(n, volt, vref, bits):
```

©Adafruit Industries
n = simpleio.map_range(volt, 0, vref, 0, bits)
pitches.append(n)

# brings values from volts.py into individual arrays
for v in volts:
    # map_volts function to map 1v/oct values to 12 bit
    # and append to pitches[]
    map_volts(v['label'], v['1vOct'], 5, 4095)
    # append midi note numbers to midi_notes[] array
    midi_notes.append(v['midi'])

while True:
    # read incoming midi messages
    msg = midi.receive()
    # if a midi msg comes in...
    if msg is not None:
        # if it's noteoff...
        if isinstance(msg, NoteOff):
            # send 0 volts on dac
            dac.raw_value = 0
            # turn off gate pin
            gate.value = False
        # if it's noteon...
        if isinstance(msg, NoteOn):
            # compare incoming note number to midi_notes[]
            z = midi_notes.index(msg.note)
            # limit note range to defined notes in volts.py
            if msg.note < 36:
                msg.note = 36
            if msg.note > 96:
                msg.note = 96
            # send corresponding 1v/oct value
            dac.raw_value = int(pitches[z])
            # turn on gate pin
            gate.value = True

Upload the Code and Libraries to the QT Py RP2040

After downloading the Project Bundle, plug your QT Py RP2040 into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called CIRCUITPY. Unzip the folder and copy the following items to the QT Py RP2040’s CIRCUITPY drive.

- lib folder
- volts.py
- code.py

Your QT Py RP2040 CIRCUITPY drive should look like this after copying the lib folder, volts.py file and the code.py file.
volts.py File

The volts.py file is a helper file that contains a dictionary (volts[]) with the phenetic note name, the MIDI note number and the 1V/oct voltage. The range of notes are C2 to C7, or 0V to 5V.

```python
volts = [
    {'label':"C-2","midi":36,'1vOct':0.000},
    {'label':"C♯2","midi":37,'1vOct':0.083},
    {'label':"D-2","midi":38,'1vOct':0.167},
    ...
    {'label':"B-6","midi":95,'1vOct':4.917},
    {'label':"C-7","midi":96,'1vOct':5.000},
]
```

How the CircuitPython Code Works

First, USB MIDI, the digital output for the gate signal, I2C and the DAC are setup. Note that board.I2C() is being used rather than STEMMA_I2C().

```python
# midi channel setup
midi_in_channel = 1
midi_out_channel = 1

# USB midi setup
midi = adafruit_midi.MIDI(
    midi_in=usb_midi.ports[0], in_channel=0, midi_out=usb_midi.ports[1],
    out_channel=0
)

gate = DigitalInOut(board.A1)
gate.direction = Direction.OUTPUT

dac = adafruit_mcp4725.MCP4725(i2c)
dac.raw_value = 4095
```

©Adafruit Industries
Mapping Voltages

Two arrays are created: `midi_notes[]` and `pitches[]`. `midi_notes[]` will hold the MIDI note numbers as defined in volts.py. `pitches[]` will hold the 12-bit DAC values that correspond with the 1V/oct values in volts.py.

```python
# array for midi note numbers
midi_notes = []
# array for 12 bit 1v/oct values
pitches = []
```

The `map_volts()` function, uses simpleio's `map_range()` function to map the 1V/oct values from volts.py to 12-bit values for the MCP4725 DAC. These 12-bit values are then added to the `pitches[]` array.

```python
# function to map 1v/oct voltages to 12 bit values
# these values are added to the pitches[] array
def map_volts(n, volt, vref, bits):
    n = simpleio.map_range(volt, 0, vref, 0, bits)
pitches.append(n)
```

A `for` statement iterates through the dictionary in volts.py and uses `map_volts()` to map the 1V/oct values to 12-bit values. Then, the MIDI note numbers in volts.py are added to the `midi_notes[]` array.

The indexes of each array correspond with each other for the same note and will be used in the loop. For example, `midi_notes[0]` equals MIDI note 36 and `pitches[0]` equals 0, which are two different ways of saying note C2.

```python
# brings values from volts.py into individual arrays
for v in volts:
    # map_volts function to map 1v/oct values to 12 bit
    # and append to pitches[]
    map_volts(v['label'], v['1vOct'], 5, 4095)
    # append midi note numbers to midi_notes[] array
    midi_notes.append(v['midi'])
```

The Loop

The loop begins by listening to incoming MIDI messages. In this instance of the code, the specific messages being listened for are `NoteOn` and `NoteOff` messages.

If a `NoteOn` message is received, the message's note number is checked against the `midi_notes` array and the matching index is defined as `z`. The DAC's value is set to `pitches[z]`, sending the corresponding 1V/oct voltage as a 12-bit value. `gate`'s value is set to `True`. 
If a `NoteOff` message is received, the DAC’s value is set to 0 and `gate`’s value is set to `False`.

```python
while True:
    # read incoming midi messages
    msg = midi.receive()
    # if a midi msg comes in...
    if msg is not None:
        # if it's noteoff...
        if isinstance(msg, NoteOff):
            # send 0 volts on dac
            dac.raw_value = 0
            # turn off gate pin
            gate.value = False
        # if it's noteon...
        if isinstance(msg, NoteOn):
            # compare incoming note number to midi_notes[]
            z = midi_notes.index(msg.note)
            # limit note range to defined notes in volts.py
            if msg.note < 36:
                msg.note = 36
            if msg.note > 96:
                msg.note = 96
            # send corresponding 1v/oct value
            dac.raw_value = int(pitches[z])
            # turn on gate pin
            gate.value = True
```

Wiring
Components

Solder short plug headers to the QT Py RP2040.

Solder short socket headers to the 1/4 perma proto board. Solder one stack of headers in column D, rows 9-15. Solder the other stack in column H, rows 9-15.

Solder one audio jack's sleeve pin to column C, row 6 and its tip to column C, row 2.

Solder the second audio jack's sleeve pin to column H, row 6 and its tip to column H, row 2.
Ground Rail

Solder one wire from the ground rail to column A, row 14. This will connect the QT Py RP2040's GND connection to the ground rail.

Solder one wire from the ground rail to column A, row 6. Then, a second wire from column E, row 6 to column F, row 6. This connects the audio jacks' sleeve pins to GND.

Power Rail

Solder one wire from the power rail to column A, row 15 (red wire). This connects the QT Py RP2040's 5V pin to the power rail.

Gate Out Signal

Solder a wire from column J, row 2 to column J, row 14 (green wire). This connects the gate signal audio jack's tip to the QT Py RP2040's pin A1.
The MCP4725 DAC

The MCP4725 DAC is wired to the 1/4 perma proto via its header pins. You can use the bottom plate as a guide for how long the wires need to be.

DAC Power

Solder the DAC's VIN pin to the power rail (red wire). This allows the DAC to have a 0 to 5V range.

DAC Ground

Solder the DAC's GND pin to the ground rail (black wire).
DAC I²C

Solder the DAC's SCL pin to column I, row 10 (yellow wire).

Solder the DAC's SDA pin to column J, row 11 (blue wire).

DAC Output

Solder the DAC's VOUT pin to the left audio jack's tip at column A, row 2 (white wire).

And that completes the wiring!
Assembly
Prepare the Skull

Attach five 12mm M2.5 stand-offs with M2.5 screws into the skull's mounting holes in its nose and mouth.
Prepare the Boards

Attach an 8mm M2.5 stand-off with an M2.5 screw into the 1/4 perma proto's mounting hole located between the socket headers.

Then, plug the QT Py RP2040 into the socket headers.
Attach the Boards

Slot the 1/4 perma proto and MCP4725 STEMMA board onto the mounted stand-offs. The two audio jacks will slot into the skull's eyes.

Secure the boards to the stand-offs with five 8mm M2.5 stand-offs.

Finish

Secure the back plate to the stand-offs with M2.5 screws. That completes the assembly!
Usage

Connect the QT Py RP2040 via USB to a USB MIDI host or a computer running your favorite DAW (digital audio workstation).

Use the skull's left eye jack for a gate out signal. Use the skull's right eye jack for a 1V/oct output.

When you send a MIDI `NoteOn` message, a gate and 1V/oct voltage will be sent out of the audio jacks. A `NoteOff` message stops the gate signal.

You can change up the code.py file to utilize different MIDI messages or different CV values depending on your needs and synth gear.
Going Further - Winterbloom's CircuitPython Eurorack Modules

If you're interested in seeing how far you can go with Eurorack/CV controlled synths and CircuitPython, be sure to check out Winterbloom. Winterbloom is an open source synthesizer company with hardware and software designed by Thea Flowers.

She designed two modules that utilize CircuitPython: Big Honking Button and Sol. Both modules allow you to modify the CircuitPython code running onboard for custom firmware. Sol is also very similar to this skull project, since it is a MIDI to CV converter, but with four gate outputs and four CV outputs.