Grand Central USB MIDI Controller in CircuitPython

Created by John Park


Last updated on 2023-10-18 04:42:30 PM EDT
# Table of Contents

## Overview
- Parts
- Optional

## Build the MIDI CC Controller
- Go Beyond the Breadboard

## Code USB MIDI in CircuitPython
- Prepare the Grand Central
- Code
- Using Adafruit USB MIDI
- Code Walkthrough
- Libraries
- MIDI Instance
- Knob Setup
- CC Ranges
- CC Value
- Hysteresis
- Main Loop
- MIDI Message Send
- In Use
- MIDI Monitor
- Rack Patch
- Get Mobile
Overview

You can build your own 16-knob USB MIDI CC controller! Music software is great, but don't you miss having real knobs to turn? Harness the massive amounts of Grand Central M4 Express I/O by using CircuitPython to create a MIDI controller of your dreams to dial in sequencer values, DJ software mixes and effects, and any other value you like in your DAW, synthesizer, sequencer, or DJ tools!

The Grand Central can send USB MIDI messages, such as Note On and Note Off, as well as CC (continuous controller) numbers and values. This means you can adjust virtual knobs in your music software using real, physical knobs!

In this guide, we'll wire up 16 potentiometers and program the Grand Central MIDI Controller to do your knobby bidding!

You could do a smaller scale version of this project with any Adafruit M0 or M4 Express board, but you will have fewer analog inputs to work with.

Parts

1 x Adafruit Grand Central M4 Express
featuring the SAMD51  
https://www.adafruit.com/product/4064

16 x Potentiometer with Built In Knob
10K ohm  
https://www.adafruit.com/product/4133
Alternatively, you can use these trim pots, but they are harder to turn:

16 x Breadboard trim potentiometer
10K Ohm  
https://www.adafruit.com/product/356

1 x USB Patterned Fabric Cable
A/MicroB - 3ft  

1 x Premium Male/Male Jumper Wires
40 x 6” (150mm)  
https://www.adafruit.com/product/758

1 x Full sized breadboard
Gluten free  
https://www.adafruit.com/product/239

Optional

1 x Mega protoshield
for Grand Central or Arduino Mega  
https://www.adafruit.com/product/192

1 x iOS Lightning to USB OTG Cable
Adapt!  
https://www.adafruit.com/product/3940

1 x Adafruit Perma-Proto Full-sized Breadboard
Single  
https://www.adafruit.com/product/1606

1 x Hook-up Wire Spool Set - 22AWG Solid Core
10 x 25ft  
https://www.adafruit.com/product/3174

For the MEGA proto shield version, I used some vertical PC mount 9mm 100k linear pots with 6mm knobless shafts. You can them here.

One note, I cut off their mounting tabs for the tight fit.

To add additional knobs you would need an ADC expander, such as this https://www.adafruit.com/product/1083
Build the MIDI CC Controller

Here's how you'll wire up the board. It's simple, really, it just needs to be repeated a bunch of times!

To read a potentiometer, we'll connect the left leg to ground and the right leg to 3.3V. The middle leg is connected to the pot's wiper, which is a variable resistor. The entire arrangement acts as a variable voltage divider.

By connecting the middle leg to an analog input pin on the Grand Central, we can read the varying voltage level.
First add a potentiometer to the breadboard with the legs at the bottom of the switch.

Wire a black jumper from the left leg to ground.

Wire a red jumper from the right leg to power.

This leaves the center leg to be wired to the an analog input on the Grand central.

Repeat this for a total of eight pots on the top half of the breadboard.
Now, you can wire the center legs of each pot to the first eight analog inputs on the Grand Central. You will also run a black wire from the breadboard ground rail to Grand Central GND pin and a red wire from breadboard power rail to Grand Central 3.3V pin. Do not use 5V!

You need to connect the breadboard power rail to the Grand Central 3.3V line and not the 5V line. You need to wire the breadboard ground to a ground on the Grand Central.
Be sure to jumper the ground and power rails on the lower half of the breadboard to their respective rails on the top half, then add the other eight pots as shown here.

You’re ready now to prep the board and code it for use!

**Go Beyond the Breadboard**

For a more advanced build, you can go beyond the breadboard and onto a Perma Proto board, or even a [MEGA Shield](#) as seen here!
Code USB MIDI in CircuitPython

Prepare the Grand Central

We'll be using CircuitPython for this project. Are you new to using CircuitPython? No worries, there is a full getting started guide here.

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and its installation in this tutorial. Mu 1.0.2
has support for detecting the Adafruit Grand Central as a valid board - if you have an older version, please upgrade.

Follow this guide for instructions on installing the latest release version of CircuitPython for the Grand Central.

You'll also need to add the following libraries for this project. Follow this guide on adding libraries. The ones you'll need are:

- neopixel
- simpleio
- adafruit_midi

Download the latest adafruit-circuitpython-bundle .zip file as instructed in the guide linked below. Unzip the file and drag those libraries to the lib folder on your Grand Central M4 CIRCUITPY drive (create the lib directory if it does not already exist).

---

**Code**

You can now upload the code to your Grand Central so it will read the pots and send USB MIDI commands.

Here is the code we'll use. Copy it and then paste it into the Mu editor. Save it to your Grand Central M4 as code.py

```python
# SPDX-FileCopyrightText: 2023 John Park for Adafruit Industries
# SPDX-License-Identifier: MIT

# Grand Central MIDI Knobs
# for USB MIDI and optional UART MIDI
# Reads analog inputs, sends out MIDI CC values
# with Kattni Rembor and Jan Goolsbey for range and hysteresis code

import time
import board
```
import busio
from simpleio import map_range
from digitalio import DigitalInOut, Direction
import usb_midi
import adafruit_midi  # MIDI protocol encoder/decoder library
from adafruit_midi.control_change import ControlChange

# pick your USB MIDI out channel here, 1-16
MIDI_USB_channel = 1
# pick your classic MIDI channel for sending over UART serial TX/RX
CLASSIC_MIDI_channel = 2

midi_usb = adafruit_midi.MIDI(  
    midi_out=usb_midi.ports[1], out_channel=MIDI_USB_channel - 1
)
# use DIN-5 or TRS MIDI jack on TX/RX for classic MIDI
midi_uart = busio.UART(board.TX, board.RX, baudrate=31250, timeout=0.001)

led = DigitalInOut(board.D13)  # activity indicator
led.direction = Direction.OUTPUT

knob_count = 16  # Set the total number of potentiometers used

# Create the input objects list for potentiometers
knob = []
for k in range(knob_count):
    knobs = AnalogIn(getattr(board, "A{}".format(k)))  # get pin # attribute, use string formatting
    knob.append(knobs)

# assignment of knobs to cc numbers
cc_number = [
    1,  # knob 1, mod wheel
    2,  # knob 2, breath control
    7,  # knob 3, volume
    10,  # knob 4 pan
    11,  # knob 5, expression
    53,  # knob 6
    54,  # knob 7
    73,  # knob 8
    74,  # knob 9, Filter frequency cutoff
    71,  # knob 10, Filter resonance
    58,  # knob 11
    59,  # knob 12
    60,  # knob 13
    61,  # knob 14
    62,  # knob 15
    63,  # knob 16
]

# CC range list defines the characteristics of the potentiometers
# This list contains the input object, minimum value, and maximum value for each knob.
cc_range = [
    (0, 127),  # knob 0: C2 to B5: 49-note keyboard
    (0, 127),  # knob 1
    (0, 127),  # knob 2
    (0, 127),  # knob 3
]
(0, 127), # knob 4
(0, 127), # knob 5
(0, 127), # knob 6
(0, 127), # knob 7
(0, 127), # knob 8
(0, 127), # knob 9
(0, 127), # knob 10
(0, 127), # knob 11
(0, 127), # knob 12
(0, 127), # knob 13
(0, 127), # knob 14
(0, 127), # knob 15
]
print("---Grand Central MIDI Knobs---")
print(" USB MIDI channel: {}".format(MIDI_USB_channel))
print(" TRS MIDI channel: {}".format(CLASSIC_MIDI_channel))

# Initialize cc_value list with current value and offset placeholders
cc_value = []
for _ in range(knob_count):
    cc_value.append((0, 0))
last_cc_value = []
for _ in range(knob_count):
    last_cc_value.append((0, 0))

# range_index converts an analog value (ctl) to an indexed integer
# Input is masked to 8 bits to reduce noise then a scaled hysteresis offset
# is applied. The helper returns new index value (idx) and input
# hysteresis offset (offset) based on the number of control slices (ctrl_max).
def range_index(ctl, ctrl_max, old_idx, offset):
    if (ctl + offset > 65535) or (ctl + offset < 0):
        offset = 0
    idx = int(map_range((ctl + offset) & 0xFF00, 1200, 65500, 0, ctrl_max))
    if idx != old_idx:  # if index changed, adjust hysteresis offset
        offset = int(
            0.25 * sign(idx - old_idx) * (65535 / ctrl_max)
        )  # edit 0.25 to adjust slices
    return idx, offset

def sign(x):  # determine the sign of x
    if x >= 0:
        return 1
    else:
        return -1

while True:
    # read all the knob values
    for i in range(knob_count):
        cc_value[i] = range_index(  
            knob[i].value,  
            (cc_range[i][1] - cc_range[i][0] + 1),  
            cc_value[i][0],  
            cc_value[i][1],  
        )
        if cc_value[i] != last_cc_value[i]:  # only send if it changed
            # Form a MIDI CC message and send it:
            midi_usb.send(ControlChange(cc_number[i], cc_value[i][0] + cc_range[i][0])))
            classic_midi.send(  
                ControlChange(cc_number[i], cc_value[i][0] + cc_range[i][0]))  
        last_cc_value[i] = cc_value[i]
    led.value = True
Before we test it out, let's have a closer look at how it works.

**Using Adafruit USB MIDI**

The `adafruit_midi` library code for CircuitPython allows you to easily send the most commonly used MIDI messages.

### Create the MIDI Object

The MIDI object can be instantiated this way:

```python
usb_midi = adafruit_midi.MIDI(midi_out=usb_midi.ports[1],
out_channel=USB_MIDI_channel - 1)
```

This allows us to refer to it in the code with the nice, short name `midi`. And we're also using the `out_channel` argument (which is zero indexed) to set the outgoing MIDI channel through which messages will be sent -- in this case, MIDI channel 1. The possible range is 0-15, which correlates to MIDI channels 1-16.

Then, to send messages, we can use four types:

- Note On
- Note Off
- Control Change (a.k.a., Continuous Controller or CC)
- Pitch bend

#### Note On

`note_on` is used to send a MIDI Note On message. First argument is the note number, **0-127**. Second argument is the velocity, **0-127**. Typical 88-key piano note range is 36-108 which correlate to pitches C2 to C8.

Example:

- `usb_midi.send(NoteOn(60, 64))` sends a MIDI message of Note On number 60 (C3 on the keyboard) at a velocity of 64
Note Off

`note_off` is used to send a MIDI Note Off message. First argument is the note number, **0-127**. Second argument is the velocity, **0-127**.

Example:

- `usb_midi.send(NoteOff(60, 64))` sends a MIDI message of Note Off number 60, at a velocity of 64. (The velocity doesn't actually matter in most cases, but can be used for interesting effects with harpsichords and other plucked instruments.)

Pitch Bend

`pitch_bend` sends a MIDI Pitch Wheel message. Range is **0-16383**. A value of 8192 equates to no pitch bend. A value > **8192** is an upward bend, while a value < **8192** is a negative pitch bend.

Example:

- `usb_midi.send(PitchBend(10000))` sends and increasing pitch bend, in this case a value of **10000**

Control Change (CC)

`control_change` sends a MIDI CC ('control change' or 'continuous controller') message. First argument is the controller number, **0-15**. Second argument is the control value, **0-127**.

Example:

- `usb_midi.send(ControlChange(4, 100))` sends a MIDI control change on control number 4 with a value of **100**.

This is a good resource () for greater details on the MIDI protocol.
Each MIDI channel can use up to sixteen CC controller numbers. In software, these are usually assignable to anything you like. Here's a table of typical uses, particularly on hardware synthesizers and other MIDI gear:

- 0 Bank Select
- 1 Modulation Wheel
- 2 Breath Controller
- 3 Undefined
- 4 Foot Controller
- 5 Portamento time
- 6 Data Entry Most Significant Bits
- 7 Volume
- 8 Balance
- 9 Undefined
- 10 Pan
- 11 Expression
- 12 Effect Controller 1
- 13 Effect Controller 2
- 14 Undefined
- 15 Undefined

The good news is, you can pretty much ignore these crusty old standards in your software and map any knob to any function! So, once you're inside your favorite software, you'll pick a software knob, enter MIDI learn mode, and assign one of your sixteen Grand Central knobs to do the job!

## Code Walkthrough

### Libraries

First, we'll import the libraries:

```python
import time
import board
import busio
from simpleio import map_range
from analogio import AnalogIn
from digitalio import DigitalInOut, Direction
import usb_midi
import adafruit_midi
from adafruit_midi.control_change import ControlChange
```
**MIDI Instance**

**USB MIDI**

Then, we'll define the `adafruit_midi` instance and tell it which MIDI channel to use. MIDI channel number 1 is a good default, unless you have something else plugged into your computer already using it.

```python
usb_midi = adafruit_midi.MIDI(midi_out=usb_midi.ports[1],
                             out_channel=USB_MIDI_channel - 1)  # (Remember, this is zero indexed, so a 1 is subtracted)
```

**Classic MIDI**

With a similar setup, we'll use the TX/RX UART serial port with a TRS jack or DIN-5 connector to send classic MIDI:

```python
uart = busio.UART(board.TX, board.RX, baudrate=31250, timeout=0.001)
# initialize UART
classic_midi = adafruit_midi.MIDI(
                              midi_out=uart, midi_in=uart, out_channel=CLASSIC_MIDI_channel - 1, debug=False)
```

**Knob Setup**

Next, we'll set up the knob inputs. We'll use a variable to define the number of knobs used, which makes it simple to go with fewer than the max of 16 if needed.

```python
knob_count = 16
```

We'll then define the list of knobs with easy to use names that are actually pointing at the AnalogIn pins. We could do something like this 16 times:

```python
knob0 = AnalogIn(board.A0)
```

but it's neater to wrap it up into a loop like this:

```python
knob = []
for k in range(knob_count):
    knob.append(knob0)
```
knobs = AnalogIn(getattr(board, "A{}".format(k)))
knob.append(knobs)

CC Ranges

While the MIDI CC value range runs from 0 to 127, in some cases we'll want to output only a subset of that range when we turn a knob fully. For example, 0 to 127 is great if you're controlling a mixer knob, but it's too huge of a range if you're sending out chromatic pitch values. Typical 88-key MIDI keyboards range from 21 (a very low A0) to 108 (a super high C8).

When using a knob for sequencing melodies, you'll probably want an even tighter range, such as 36 (C2) to 84 (B5) or smaller.

So, we will create a list of ranges that can be adjusted in code per knob:

```python
cc_range = [
    (36, 84),  # knob 0: C2 to B5: 49-note keyboard
    (36, 84),  # knob 1
    (36, 84),  # knob 2
    (36, 84),  # knob 3
    (36, 84),  # knob 4
    (36, 84),  # knob 5
    (36, 84),  # knob 6
    (36, 84),  # knob 7
    (0, 127),  # knob 8: 0 to 127: full range MIDI CC/control voltage for VCV Rack
    (0, 127),  # knob 9
    (0, 127),  # knob 10
    (0, 127),  # knob 11
    (0, 127),  # knob 12
    (0, 127),  # knob 13
    (0, 127),  # knob 14
    (0, 127),  # knob 15
]
```

CC Value

We will create a variable list to store the value of each knob:

```python
cc_value = []
for c in range(knob_count):
    cc_value.append((0,0))
```

Hysteresis

This helper function, created by Jan Goolsbey, is used to reduce value jitter when a potentiometer is right on the edge between two values:
def range_index(ctl, ctrl_max, old_idx, offset):
    if (ctl + offset &gt; 65535) or (ctl + offset &lt; 0):
        offset = 0
    idx = int(map_range((ctl + offset) &amp; 0xFF00, 1200, 65500, 0, ctrl_max))
    if idx != old_idx:  # if index changed, adjust hysteresis offset and set flag
        # offset is 25% of the control slice (65536/ctrl_max)
        offset = int(0.25 * sign(idx - old_idx) * (65535 / ctrl_max))  # edit 0.25
to adjust slices
    return idx, offset

The next function is a helper used along with the range_index function to determine
the direction of the potentiometer's movement:

def sign(x):  # determine the sign of x
    if x &gt;= 0:
        return 1
    else:
        return -1

Main Loop

Now, in the main loop of the program we'll do these things:

- Read the knobs
- Adjust their values to conform to the range table and reduce jitter
- Send their values out as the properly formed USB MIDI messages

Here is the loop that checks and adjust the knob values:

while True:
    # read all the knob values
    for i in range(knob_count):
        cc_value[i] = range_index(knob[i].value,
MIDI Message Send

And finally, the thing we've been waiting for -- sending the message, but only if it has changed since the last check.

```python
for i in range(knob_count):
    cc_value[i] = range_index(
        knob[i].value,
        (cc_range[i][1] - cc_range[i][0] + 1),
        cc_value[i][0],
        cc_value[i][1],
    )
    if cc_value[i] != last_cc_value[i]:  # only send if it changed
        # Form a MIDI CC message and send it:
        usb_midi.send(ControlChange(cc_number[i], cc_value[i][0] + cc_range[i][0]))
        classic_midi.send(
            ControlChange(cc_number[i], cc_value[i][0] + cc_range[i][0])
        )
        last_cc_value[i] = cc_value[i]
led.value = True
```

In Use

Now, it's time to use your Grand Central USB MIDI Knob Controller! With it plugged into your computer over USB, launch a DAW, software synthesizer/sequencer, or DJ tool. Here are some examples of free, open source synths for Linux, Windows, and mac os:

- Helm
- VCV Rack
- Pure Data
- Ardour

This page shows more details on using a MIDI controller with Helm.
MIDI Monitor

You can use this handy Chrome browser MIDI Monitor web app () to simply read the values of your Grand Central controller.

Rack Patch

This is an example of a patch made in VCV Rack, the open source modular software synthesizer:
The module in the upper left corner, MIDI-CC, is used to connect the Grand Central MIDI Knob controller to the rest of the modules. You can see 16 patch cable running from it to an 8-step pitch sequencer, as well as various other modules to control the envelope and filter of the sound. All with your real, physical knobs!

Get Mobile

The Grand Central USB MIDI Knob Controller also works great for controlling many iOS apps that have tiny virtual knobs, for example [AudioKit Synth One](https://www.audiokit.io/synth-one) (iPad only).

You'll need to check if your app supports MIDI controller assignments and use an [OTG cable](https://www.adafruit.com/product/345) to plug in the Grand Central.

Of course, you can also use the Grand Central with Ableton Live, Propellerhead Reason, FL Studio, Logic, Traktor, Max/MSP, and other professional apps!