Trellis Feather DSP-G1 Synthesizer

Created by John Park

https://learn.adafruit.com/feather-trellis-dsp-g1-synthesizer

Last updated on 2023-08-29 03:41:56 PM EDT
# Table of Contents

**Overview**
- Parts
- Material & Tools

**Build the Feather Synth Circuit**
- Synth Chip Circuit
- Feather Headers
- Breadboarding
- Solder the Headers
- IC Socket
- Wire Routing
- Passive Components
- Headphone Jack
- Synth Chip

**Code Test**
- Setup
- MIDI Library
- MIDI Messages
- Code Demo
- Sweeps

**Add Trellis Keypads**
- Keypad Input
- Build the Double Trellis
- Wiring the Trellis Boards
- Power Connection
- Tidy Wiring
- Board Connections
- Testing the Trellis Boards

**Knobs Knobs Knobs**
- Wiring Connector
- Parts
- Housing Connection
- Right Angle Header Pins
- Potentiometer Connections
- Ground and Power Sharing

**Selector Switch**

**Reset and Power**
- Reset Button
- Button Prep
- Wire the Reset Button
- Power Switch
- Activity LED Ring
- LED Ring Wiring
- Plug in the LED Ring
Case Construction
  • Assembly

Code the Synthesizer
  • Logical Pads
  • Synthesizer Parameters
  • Loop
  • Knob Readings
  • Code Download

Play Music
Build your own synthesizer with keypad note inputs and knobs for parameter controls!
At the heart of this instrument is the DSP-G1 MIDI synthesizer voice chip — an analog modeling synth by Jan Ostman that can read in MIDI data for notes and control change (CC) messages, and outputs sound to play over any powered speakers, headphones, or amplifier system!

The DSP-G1 makes beautiful, classic paraphonic synth sounds, similar to the famous 1980s Roland Juno-6. You can play up to five notes at once, each with three oscillators, and you can dial in multiple wave shapes, detuning, LFO, 24db filter, ADSR envelopes for the amplifier and filter, and more!
The Feather M0 Express can read your Trellis keypad buttons and potentiometer knobs and then send these commands to the synth chip as MIDI messages.

The Trellis Feather DSP-G1 synthesizer code is configured to play eight note scales in three octaves, in both momentary and held (latching) modes, with the bottom row of eight keypads reserved for input functions. But, there’s nothing to stop you from turning it into a step sequencer, random note generator, chording machine, or who knows what!

If you’d like to use Trellis keypads to send MIDI through USB or Bluetooth to your computer, search for “MIDI” in the Learn Guide search bar!

You can watch the build from the livestream here:
Parts

Listed below are the parts you use to build your synthesizer.

1 x **Feather M0 Express**
Designed for CircuitPython ATSAMD21 Cortex M0
https://adafru.it/3403

1 x **DSP-G1**
MIDI Synth Voice Chip
https://adafru.it/2400

1 x **8 Pin IC Socket socket**
for 8-pin 0.3" Chips
https://adafru.it/2202

1 x **Adafruit Perma-Proto**
1/2 Sized Breadboard PCB
https://adafru.it/1609

1 x **Short Feather Headers Kit - Male**
12-pin and 16-pin Set
https://adafru.it/3002

1 x **Short Feather Headers Kit - Female**
12-pin and 16-pin Set
https://adafru.it/2940

1 x **Right angle male header pins 10 pack**
Break-away 36-pin strip (10 pack)
https://adafru.it/1540

1 x **Panel Mount Stereo Audio Extension Cable**
1/8" / 3.5mm TRS
https://adafru.it/3319

1 x **Breadboard-Friendly 3.5mm Headphone jack**
1 x **10uF Electrolytic Capacitors**
Pack of 10
[https://adafru.it/2195](https://adafru.it/2195)

1 x **0.1uF Ceramic Capacitors**
10 pack
[https://adafru.it/753](https://adafru.it/753)

1 x **Through-Hole Resistors 470 ohm**
Pack of 25
[https://adafru.it/2781](https://adafru.it/2781)

6 x **Panel Mount 10k Potentiometer**
Breadboard Friendly 10k Linear
[https://adafru.it/562](https://adafru.it/562)

6 x **Potentiometer Knob**
Soft Touch T18 White
[https://adafru.it/2047](https://adafru.it/2047)

2 x **Adafruit Trellis Monochrome Driver PCB**
for 4x4 Keypad and 3mm LEDs
[https://adafru.it/1616](https://adafru.it/1616)

2 x **Silicone Elastomer 4x4 Button Keypad**
for 3mm LEDs
[https://adafru.it/1611](https://adafru.it/1611)

2 x **Diffuse Red 3mm LEDs**
25 pack
[https://adafru.it/777](https://adafru.it/777)

1 x **Rugged Metal Pushbutton with White LED Ring**
16mm White Momentary
[https://adafru.it/558](https://adafru.it/558)

1 x **Mini 8-way Rotary Selector Switch**
SP8T
[https://adafru.it/2925](https://adafru.it/2925)

1 x **Slim Metal Knob**
10mm dia. x 10mm T18
[https://adafru.it/2058](https://adafru.it/2058)

1 x **Silicone Cover Stranded-Core Wire**
Black
[https://adafru.it/1881](https://adafru.it/1881)

1 x **Silicone Cover Stranded-Core Wire**
Red
[https://adafru.it/1877](https://adafru.it/1877)

1 x **Silicone Cover Stranded-Core Wire**
Yellow
[https://adafru.it/1879](https://adafru.it/1879)
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 x Silicone Cover Stranded-Core Wire Green</td>
<td></td>
<td><a href="https://adafru.it/1880">https://adafru.it/1880</a></td>
</tr>
<tr>
<td>1 x Multi-Colored Heat Shrink Pack 3/32” + 1/8” + 3/16” Diameters</td>
<td></td>
<td><a href="https://adafru.it/1649">https://adafru.it/1649</a></td>
</tr>
<tr>
<td>1 x Large Single Row Housing Pack for DIY Jumper Cables</td>
<td></td>
<td><a href="https://adafru.it/3146">https://adafru.it/3146</a></td>
</tr>
<tr>
<td>1 x Small Single Row Wire Housing Pack for DIY Jumper Cables</td>
<td></td>
<td><a href="https://adafru.it/3145">https://adafru.it/3145</a></td>
</tr>
<tr>
<td>1 x Raw Custom Jumper Wires Female/Female 40 x 6”</td>
<td></td>
<td><a href="https://adafru.it/3141">https://adafru.it/3141</a></td>
</tr>
<tr>
<td>1 x Raw Custom Jumper Wires Male/Male 40 x 6”</td>
<td></td>
<td><a href="https://adafru.it/3142">https://adafru.it/3142</a></td>
</tr>
<tr>
<td>1 x Raw Custom Jumper Wires Female/Male 40 x 6”</td>
<td></td>
<td><a href="https://adafru.it/3663">https://adafru.it/3663</a></td>
</tr>
<tr>
<td>1 x Hook-up Wire Spool Set 22AWG Solid Core 10 x 25ft</td>
<td></td>
<td><a href="https://www.adafruit.com/product/3174">https://www.adafruit.com/product/3174</a></td>
</tr>
<tr>
<td>1 x Panel Mount Extension USB Cable Micro B Male to Micro B Female</td>
<td></td>
<td><a href="https://adafru.it/3258">https://adafru.it/3258</a></td>
</tr>
<tr>
<td>1 x USB Cable A/MicroB - 3ft</td>
<td></td>
<td><a href="https://adafru.it/592">https://adafru.it/592</a></td>
</tr>
<tr>
<td>1 x 5V 1A USB port power supply UL Listed</td>
<td></td>
<td><a href="https://adafru.it/501">https://adafru.it/501</a></td>
</tr>
<tr>
<td>1 x Half-size Breadboard optional</td>
<td></td>
<td><a href="https://www.adafruit.com/product/64">https://www.adafruit.com/product/64</a></td>
</tr>
<tr>
<td>1 x Stickvise PCB Vise optional</td>
<td></td>
<td><a href="https://adafru.it/3197">https://adafru.it/3197</a></td>
</tr>
<tr>
<td><strong>Material &amp; Tools</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In addition to the parts above, you'll also want a cardboard box, roughly 9&quot;x6&quot;x3&quot; in size to use as an enclosure.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tools required are a soldering iron with solder, box cutter or hobby knife, diagonal cutters, and a pair of wire strippers. You may already have these, but if not, here are some suggested tools:

1 x **Atten 50-Watt Soldering Iron**  
SA-50  
[https://www.adafruit.com/product/2163](https://www.adafruit.com/product/2163)

1 x **Hakko Wire Strippers**  
20-30 AWG  
[https://adafru.it/527](https://adafru.it/527)

1 x **Flush Diagonal Cutters**  
CHP170  
[https://adafru.it/152](https://adafru.it/152)
We'll start with the heart of the project — the DSP-G1 synth chip and the Feather M0 Express to send it MIDI commands. On its own, the synth chip can receive MIDI messages from any device that can send them, such as a MIDI keyboard or sequencer. But, the addition of the Feather M0 Express allows us to use a wide array of sensors, buttons, knobs, and more as input devices. The Feather can be programmed to read input devices and then output MIDI messages to the synth chip.

You could even code and arpeggiator, sequencer, or MIDI song player with the Feather and DSP-G1 combo!

Download the application manual for the DSP-G1 for more details.

**Synth Chip Circuit**

This is the synth chip's schematic, including a resistor-capacitor (RC) circuit to smooth the signal to the audio output. Instead of a typical MIDI input circuit connected to the DSP-G1's MIDI in leg, we can see the Feather M0 Express will send MIDI messages from its TX pin to the chip.
It's a good idea to test the circuit on a breadboard before committing to soldering it onto a Perma-Proto board.

**Feather Headers**

First, we'll need to solder the short male header pins to the Feather M0 Express so that it can be attached to a breadboard for testing and later to the Perma-Proto board.

Follow this guide for detailed instructions on soldering in the male header pins.
Breadboarding

If you would like to breadboard the circuit for testing before moving it to the Perma-Proto board, follow this diagram, then jump to the coding test on the next page to try it out. Once you’re satisfied everything is in working order you can return here to assemble the circuit onto the Perma-Proto for greater permanence!
Solder the Headers

Instead of soldering the Feather directly to the board, we'll used female headers. This allows the Feather to be easily removed during assembly, or even swapped out at a later date for a different one.

Use the Feather M0 Express to properly align the female headers on the Perma-Proto board as shown here
Use tape or poster putty to prevent the headers from falling out when you turn the board over
Solder all of the pins in carefully
IC Socket

Solder in the IC socket as shown, so that the DSP-G1 chip can be inserted later. Follow the column numbers and row letters as seen in the picture. Be mindful of the half circle mark at the top of the socket – it must be oriented with the mark on the left side as shown here. This will help us orient the chip properly – the dot in the upper left corner of the chip indicates the first leg, and will be placed nearest the half-circle mark of the socket.
Wire Routing

Follow these pictures to wire the chip's power, ground, and RX legs to the proper output pins of the Feather.

Be sure to clip off any excess wire after soldering each point.

Note the tiny wire used to jumper legs 5 and 6.
Add the resistor and both capacitors as shown here, soldering each in place, and clipping their excess leads. Note the polarity of the 10uF electrolytic capacitor -- make sure the negative side with the white stripe is on the right side, with its leg in point F 26. Run short lengths of hook-up wire to the positions as shown in order to connect the chip's audio output and ground to the 3.5mm stereo out jack.
Headphone Jack

The breadboard-friendly headphone jack functions the same as any 1/8' / 3.5mm TRS (tip-ring-sleeve) stereo audio jack. We can use it to listen to the synth on headphones, or connect to any amplified speaker.

Insert the jack as shown, then solder it in place.

Synth Chip

Carefully insert the DSP-G1 chip into the IC socket with the dot and half-circle marks oriented as shown— you may need to bend the legs inward slightly first for a good, clean fit. Once you're sure the legs are all aligned well in the socket, gently push the chip down until it is snuggly nestled in place.
You can now re-insert the Feather M0 Express.
You've assembled the core circuit, next we'll run a simple software test.
Setup

Let’s make some music to test things out! First, we’ll need to set up our coding environment — we’ll use Arduino and some specific libraries, but in the future you’ll be able to use CircuitPython for this project as well.

First, make sure you’re set up with Arduino IDE and able to upload a sketch to the board. You can follow this guide to get that going. Be sure to follow the directions on adding the Adafruit Feather M0 Express board definition to your Arduino IDE. Then, upload the blink sketch as a test using these instructions. When you’re ready, return here!

MIDI Library

Next, we'll install the MIDI library from Fourty Seven Effects. In the Arduino IDE click Sketch > Include Library > Manage Libraries...

Then, in the library manager, click in the search box and type MIDI synth this will narrow down the library list to just the one we want.
Click on the MIDI Library selection and then click the Install button. (It is greyed out here because I already have it installed.)

**MIDI Messages**

To tell the DSP-G1 what to do, we need to have the Feather send MIDI commands to the synth chip. MIDI (Musical Instrument Digital Interface) is a standard used to send messages between music controllers, computers, sequencers, synthesizers, and other devices.

For a great introduction to MIDI, check out this guide [Collin’s Lab: MIDI](#).

The DSP-G1's firmware is written to receive messages on MIDI channel 1 (the MIDI standard allows 16 different channels to be used in order to accommodate multiple devices in a single, interconnected system). We can send it two types of commands: Notes and Control Changes (CC) messages.

Note commands can be either a Note On or Note Off message, along with a specification for which musical note to play. The DSP-G1 is a 5-voice paraphonic synthesizer, which means we can play up to five notes at once. The range for notes is 0-127, with 64 being a middle C.

CC messages are used to control all other parameters on the DSP-G1, such as oscillator waveform range, filter cutoff frequency, and low frequency oscillator (LFO) rate. The range for these CC messages is 0-127. Later we’ll take a closer look at all of the available CC parameters on the synth.

Let's look at some sample code that will play a few notes and sweep through some CC values.
Code Demo

Copy this code, paste it into a new Arduino sketch, and then upload it to your Feather M0 Express.

```
// SPDX-FileCopyrightText: 2018 John Park for Adafruit Industries
// SPDX-License-Identifier: MIT

//DSP-G1_Synth_Parameters_Demo
//Feather M0 Express connected to DSP-G1 voice chip over TX pin

#include <MIDI.h>
MIDI_CREATE_DEFAULT_INSTANCE();
static const unsigned LED = 13;      // LED pin on the Feather

void setup() {
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
  MIDI.begin(1);

  /////////////DSP-G1 CC Parameter Settings
  //arguments are CC number, followed by value, and MIDI out channel
  MIDI.sendControlChange( 7, 120, 1);   //volume
  MIDI.sendControlChange( 1,   0, 1);   //LFO mod
  MIDI.sendControlChange(16,   6, 1);   //LFO rate
  MIDI.sendControlChange(20,   0, 1);   //LFO waveform 0-63 sine, 64-127 S/H
  MIDI.sendControlChange(74, 80, 1);    //DC Filter cutoff - higher number lets
  more harmonics through
  MIDI.sendControlChange(71, 0, 1);     //DC Filter resonance
  MIDI.sendControlChange(82, 32, 1);    //DC Filter envelope Attack
  MIDI.sendControlChange(83, 38, 1);    //DC Filter envelope Decay
  MIDI.sendControlChange(28, 64, 1);    //DC Filter envelope Sustain
  MIDI.sendControlChange(29, 32, 1);    //DC Filter envelope Release
  MIDI.sendControlChange(73, 5, 1);     //DC Oscillator envelope Attack
  MIDI.sendControlChange(75, 12, 1);    //DC Oscillator envelope Decay
  MIDI.sendControlChange(31, 60, 1);    //DC Oscillator envelope Sustain
  MIDI.sendControlChange(72, 80, 1);    //DC Oscillator envelope Release
  // Wavforms: 0 tri, 25 squarish, 50 pulse, 75 other squarish, 100 saw
}

void loop() {
  digitalWrite(LED, HIGH);
  Serial.println("Playing notes");
  //Play a C
  MIDI.sendNoteOn(24,127,1); //note 24 is C1, velocity 127, channel 1
  //Velocity isn't implemented, but it's a good habit to specify it
  delay(1000);

  //Play an E
  MIDI.sendNoteOff(24,0,1); // note 24, velocity 0, channel 1
  delay(250);
  MIDI.sendNoteOn(28,127,1); //note E1
  delay(1000);

  //Play a G
  MIDI.sendNoteOff(28,0,1);
  delay(250);
  MIDI.sendNoteOn(31,127,1); //note G1
  delay(1000);
```

©Adafruit Industries
//Play an A#
MIDI.sendNoteOff(31,0,1);
delay(250);
MIDI.sendNoteOn(34,127,1); //note G1
delay(1000);
MIDI.sendNoteOff(34,0,1);

//rest
delay(500);
//chord
MIDI.sendNoteOn(12,127,1); //C0
MIDI.sendNoteOn(28,127,1); //E2
MIDI.sendNoteOn(31,127,1); //G2
MIDI.sendNoteOn(34,127,1); //A#2
MIDI.sendNoteOn(48,127,1); //C3
//hold
delay(4000);

//filter cutoff frequency sweep
sweepFilterCutoff(1,15); //turn the filter cutoff knob to the right
sweepFilterCutoff(0,15); //turn the filter cutoff knob to the left

//filter cutoff and resonance sweeps
sweepFilterCutoff(1,15); //turn the filter cutoff knob to the right
sweepFilterResonance(1,15); //turn the filter resonance peak knob to the right
sweepFilterCutoff(0,15); //turn down the cutoff
sweepFilterResonance(0,15); //and turn down the resonance
//hold here
delay(2000);

//oscillator detune
sweepDetune(1,15); //turn up the detune
//hold to listen to that detune
delay(3000);
sweepDetune(0, 15); //turn it back down
delay(2000);

digitalWrite(LED, LOW);
Serial.println("Notes off");
MIDI.sendNoteOff(12,0,1);
MIDI.sendNoteOff(28,0,1);
MIDI.sendNoteOff(31,0,1);
MIDI.sendNoteOff(34,0,1);
MIDI.sendNoteOff(48,0,1);
delay(200);
}

void sweepFilterCutoff(int dir,int rate){ //dir 0 down, dir 1 up. rate in ms, e.g. 30
if(dir==1){ //sweep up
for(int i = 0; i < 128; i++){
  MIDI.sendControlChange(74,i,1);
delay(rate);
  Serial.print("Filter cutoff: "); Serial.println(i);
}
}
else{ //sweep down
for(int i = 127; i >=0 ; i--){
  MIDI.sendControlChange(74,i,1);
delay(rate);
  Serial.print("Filter cutoff: "); Serial.println(i);
}
}

void sweepDetune(int dir,int rate){ //dir 0 down, dir 1 up. rate in ms, e.g. 30
if(dir==1){ //sweep up
for(int i = 0; i < 128; i++){
  MIDI.sendControlChange(74,i,1);
delay(rate);
  Serial.print("Filter cutoff: "); Serial.println(i);
}
}
Plug in some headphones or an external powered speaker and you'll hear some music! The code plays a few notes individually, then holds a chord of five notes, and while those are held it sweeps through CC values on a few parameters: Filter cutoff frequency, filter resonance, and oscillator detune. It then stops the chord and starts it all over again.

Take a look at the code and the comments to see how this works. The most important commands are:

```plaintext
MIDI.sendNoteOn(note number, velocity, MIDI Out channel)
```

```plaintext
MIDI.sendNoteOff(note number, velocity, MIDI Out channel)
```

```plaintext
MIDI.sendControlChange(CC number, value, MIDI Out channel)
```

With the MIDI library in place, instructing the Feather to do a `MIDI.sendNoteOn(64, 127, 1);` will send the MIDI message over its TX pin to the DSP-G1's MIDI In leg that instructs the synth chip to play a middle C (note 64) at full key velocity (velocity is not implemented on the DSP-G1, so this number doesn't matter in this case) on MIDI channel 1.
Sweeps

I've also written some simple procedures you can see at the bottom of the code that can be used to simulate turning a knob up and down. Until we plug in potentiometers, this is a good way to simulate knob twisting and hear the effects they have on the sound.

In this code snippet you can see how the `sweepFilterCutoff()` procedure works. It takes two arguments, `dir` and `rate`, and uses those to increment or de-increment the value of the MIDI CC 74, which is assigned to filter cutoff frequency on the DSP-G1. The rate parameter determines how quickly it sweeps through the values.

```c
void sweepFilterCutoff(int dir, int rate){ //dir 0 down, dir 1 up. rate in ms, e.g. 30
    if(dir==1){ //sweep up
        for(int i = 0; i < 128; i++){
            MIDI.sendControlChange(74,i,1);
            delay(rate);
            Serial.print("Filter cutoff: "); Serial.println(i);
        }
    }
    else{ //sweep down
        for(int i = 127; i >=0 ; i--){
            MIDI.sendControlChange(74,i,1);
            delay(rate);
            Serial.print("Filter cutoff: "); Serial.println(i);
        }
    }
}
```

You can look at the full list of the DSP-G1 parameters where they are initialized at the top of the code and create your own sweep procedures to test those out as well!

Now, let's start add real input controls to the synthesizer!
Add Trellis Keypads

Keypad Input

Many synthesizers use a traditional piano style keyboard as input, but there are many other ways to play a synth! Ribbon controllers, Theremin-style antenna, touch sensors, light/distance sensors, bend sensors, button pads, knobs, and more are among the many, many possibilities when it comes to triggering and adjusting electronic instruments.

We'll create a 32 keypad input controller using two Adafruit Trellis boards. The Trellis is a wonderful input device for a few reasons:

- It has a neutral grid arrangement that doesn’t imply a particular scale mode in the same way a piano keyboard does
- The lighted button pads that can be used to convey information, such as related notes/chords, or even sequencer steps and tempo indicators
- Its buttons and lights are multiplexed and run on the I2C protocol so it only uses two input pins on the Feather, not 64 of them as individually wired buttons and LEDs would! (Except we’d run out of pins at around a dozen...)
- Up to eight Trellis boards can be combined on a single I2C bus

You can find out a lot more info on the Trellis by heading to this learn guide.
Build the Double Trellis

You can follow the Trellis guide for detailed build instructions. This video will show you how I built the particular one for this project.

Be sure to solder the A0 jumper on the second board (the one on the left when viewed from the bottom side (the side without the LEDs poking up). This gives it a unique address on I2C.

Wiring the Trellis Boards

Once you’ve built and connected your two Trellis boards, it’s time to wire them to the Perma-Proto PCB.
Cut four lengths of stranded wire, approximately 12” in length each
Strip the insulation from each end and tin the wires with solder. This makes it easier to connect them to the boards
Tin the pads on the first Trellis as shown, then solder the wires to these connections:
Red to 5V
Black to GND
Yellow to SCL
Green to SDA
We won’t be using the interrupt, so you can leave the INT pad out of it!
Power Connection

The Trellis board will connect to the 3.3v power and ground of the Feather M0 Express. To make these connections neatly, solder a jumper wire from the 3.3V row to the lower power rail on the Perma-Proto board, and another one from Feather ground to the ground rail as shown here.
The actual solder points will differ a bit from the breadboard diagram shown here. Follow the photos for the exact positions.
Tidy Wiring

There will be quite a few wires connected to this synth when we're through! In order to keep things neat, you can cut a few short lengths of heat shrink tubing to fit over the Trellis wiring bundle as shown here.
Board Connections

Now, solder the red and black wires of the Trellis to the power and ground rails respectively, as shown here.
You can remove the Feather from the PCB temporarily to solder the yellow and green wires to SCL and SDA and shown.
Testing the Trellis Boards

You can now put the Feather M0 Express back in place on the Perma-Proto board (be careful to align the pins properly), and then place the two silicone elastomer keypads on top -- note that these need to be rotated a certain way to fit the holes in the Trellis boards properly.
Follow this guide to get the Trellis library set up in Arduino. Once you have the Trellis library installed, try opening the demo code in Arduino by clicking on File > Examples > Adafruit Trellis Library > TrellisTest then upload the code to your Feather.

You'll see the lights on the first Trellis light up and you can press the buttons to turn them on and off. Success!
To test the second Trellis, simply change the I2C address in line 74 in the Arduino code from this:

```c
trellis.begin(0x70); // only one
```

to this:

```c
trellis.begin(0x71); // only one
```

You'll see the second board light up now, instead. Later, we'll use both boards at the same time.
Potentiometers are a great way to adjust the MIDI CC values on the synthesizer. By connecting a potentiometer to ground, voltage, and an analog input on the Feather, the varying positions of the potentiometer’s shaft as it is turned will be read as different voltage levels. These levels can then be mapped to a range of 0-127 values to send messages to the DSP-G1’s MIDI input leg.

We’ll start by connecting six potentiometers to the Feather. But why not connect more, since there are nineteen possible CC numbers to control on this synth? The limitation here is the number of available analog input pins on the Feather, there are six of them.

There are a few ways around this limitation — one would be to add IO expanders with more ADC (analog-to-digital) pins, such as this one (1). This is a great solution, particularly if you want one knob per parameter.

Another way to do it is to add a selector switch to allow each potentiometer to serve multiple functions. We’ll do this, and by using four different positions on the rotary selector switch, our six pots can serve as 24 different functions! This will introduce some complexity to the software, particularly around the issue of absolute knob position vs. previous parameter value when switching between bank positions. We’ll take a closer look at that in the main coding section.
Wiring Connector

To connect our potentiometers to the board, we'll make a wiring interconnect. This will allow us to easily connect and disconnect the set of pots, which can make it easier later to assemble the synth in a case.

Additionally we can reduce the amount of wiring by connecting the power and ground to the first pot and then running those in series to the remaining five. Only the middle wiper pins with the varying voltage from each of those five pot need to run all the way back to the board.
Parts

Peel of a nine wire section of raw female/female jumper wires with the colors shown. Also grab a ten position single row housing and a ten pin section of right angle male header pins.

Housing Connection

Arrange the wires as shown. Note how the brown wire is flipped over the red wire to put it at the far left. Also note the gap between the red wire and black wire. Insert the wires into the housing as shown until they click into place snugly.
Right Angle Header Pins

Insert the long ends of right angle header pin section into the cable housing as shown.

Place the short ends of the right angle header pin section into the Perma-Proto PCB so they line up with row E columns 1-10 as seen here.

Solder the header pins into place.
Now we can neatly connect or disconnect the wiring from the board!
Potentiometer Connections

Now we'll solder up all those potentiometers to the wires!
Start by sliding the black, white, and red wires onto the left, middle, and right legs of the first pot as shown and then solder them in place.

Then solder the middle leg of each of the remaining potentiometers to the gray, purple, blue, green, and yellow wires (ignore the brown wire, we'll use that later for the system reset button).

Ground and Power Sharing

Now we'll share the ground connection from the first potentiometer to the other five.
Cut five 6" lengths each of black and red silicone cover stranded wire
Solder a black wire to the first pot's ground leg (it already has an incoming black wire from the cable connected to the board)
Now, solder the other end of this wire to the ground leg of the second potentiometer
Run three more lengths like this between the ground legs of the remaining potentiometers to create a continuous series
Repeat the procedure with red wires for the power connections
Your potentiometers are wired up and ready to go! Next up, the selector switch.

Selector Switch

The rotary 8-position selector switch is just like a regular latching switch or button, except it can run to up to eight digital input pins on the Feather. We'll only be using four pins and leaving the others disconnected. By reading those pins, our software will know the position of the switch, and hence, which CC bank to use for the knob inputs.
Using similar wiring techniques you just used to connect the potentiometers to the Perma-Proto, you'll now assemble the switch wiring and interconnects. Again, note that the breadboard diagram above is set up to clearly show the connections, but the images differ to make for a neater package. Follow the images in the video slideshow here.

It doesn't matter which four pins of the selector switch you use, just keep them in a contiguous order. The center pin is for ground, which will run with a separate wire to the ground rail on the board as shown, using another right-angle header pin.

---

Reset and Power
Reset Button

It’s useful to have an easily accessible reset button for the synthesizer, especially once you build it into the case and can no longer get to the Feather’s on-board reset button. You can use the reset if you ever run into difficulties uploading new code to your synth, but it’s particularly useful in dealing with MIDI panic!

What’s MIDI panic? Most MIDI instruments have a button or key combination design to send an all notes off message just in case a glitch prevents some notes that are being played from getting the ‘off’ message in the course of regular play. They’ll drone on and on forever if you don’t have a way to resend the off message.

We’ll structure our code so that one of the first things that happens upon hitting the reset button is that an all notes off message is sent. MIDI panic managed!

We'll use this momentary button as our externally accessibly reset button. This switch can also serve double duty as a power indicator using its built-in LED light ring!
Button Prep

Peel off a brown and black raw jumper wire pair -- you can use female-male or male-male

Plug a pair of the male ends into the plastic housing -- these will plug into the reset wire and a ground wire with female jumpers

The button has five legs on it. The outer ones are for the LED ring, the inner ones are the switch contacts. C1 (common) is the common leg -- solder the black wire's raw end to C1

We want this switch to be normally open, and only close to short the Feather's reset pin when the button is pressed. Solder the brown jumper wire's raw end to the N01 (normally open) leg
Wire the Reset Button

Peel off a single black female/female raw jumper wire
Plug one end into a single hole housing
Get a two hole housing and plug in the other end of the black wire and the brown reset wire coming from the nine conductor cable you made earlier for the potentiometers as shown here
Free single end of the black wire into the right angled male header pin on the ground rail of the Perma-Proto board
Now, the reset button can be plugged into the two hole housing
Note, I plugged the reset button into the jumper housing "backwards" here, but it doesn't matter in this case if brown/black plug into/brown/black or vice versa, since either way will work and ground the reset pin when the button is pressed!
Power Switch

The Feather M0 Express makes it easy to add an on/off switch to any project. The En (enable) pin on the Feather will power down the board when connected to ground. With a single pole, dual throw (SPDT) micro switch wired to En and GND we can make a very neat power switch!

You will use two raw jumper wires (female/female or female/male will work) a three hole single row jumper housing, and a three pin right angle male header pin row to wire the SPDT micro switch to the board. Note, the three pin housing and jumper arrangement allows us to plug into the board with proper spacing, even though there are only two wires connected. You'll remove the middle pin from the header row.

Follow the images in the video slideshow below to solder the pins and jumpers. Note how we use a short jumper wire to run ground from the Perma-Proto board's ground rail to the upper part of the board where we'll plug in the switch.
Activity LED Ring

It's very helpful to have some indication that your synth (or any project for that matter!) is turned on when you flip the power switch. Instead of using a separate LED, we can used the LED ring that's integrated into our reset switch.

Note: In the breadboard diagram, this is shown as a separate LED.
LED Ring Wiring
Solder the red wire to the + leg on the button
Solder the black wire to the - leg on the button
You can use heatshrink tubing to insulate the legs and another piece to neaten the wiring
Following the images in the video slideshow, you'll see how to solder a male jumper pin to the Perma-Proto board to connect the LED ring's red wire to the Feather's pin 13, as well as a set connected to the jumpered ground row.

Plug in the LED Ring

Plug in the red LED ring wire from the button to the board's pin 13 header and the black wire to ground! We'll set up the software to light this LED when the program is running, and we'll also be able to see it blink when doing restarts for uploading code.
Case Construction
You'll want to build a case for your synthesizer next. You can get creative here and repurpose an existing enclosure, build something from scratch using wood, or even repurpose a cardboard box!

Cardboard is great to work with because it is so easy to cut and poke holes for your controls. I designed a template you can use if you want to follow this design. You can print out the template and use it to trace where to cut for the pads, knobs, switches, and ports. Or, if you have access to a laser cutter, cut out the pattern directly!
Assembly

Assemble the synthesizer into the box starting with the Trellis keypads and PCBs. You can make a small frame to hold the PCBs in place using 2.5mm nylon screws and nuts. Here is the template file for the frame, with holes that match the outer template.
Follow this video image slideshow to see how to install the remaining components.
Code the Synthesizer

The synthesizer is now ready for its full code upload! Building upon the previous two tests of code with the MIDI to synth messages and the Trellis functions, this code adds a number of other features to make the synth really fun to play!

Look at the code below and you'll see a few key features:

First, we import the MIDI and Trellis libraries as before, this time creating the Trellis object with two matrices.
Then, we create a variable `scaleMode` to select which scale we'll use. The choices are defined by MIDI note numbers and are broken up into six scales: chromatic, major, minor, dorian, mixolydian, and harmonic minor.

### Logical Pads

Next, we create the `logicalPads[]` array in order to map the physical buttons of two Trellis grids into one logical array that reads left to right from top to bottom. This makes the rest of the code much easier later on. Here you can see the mapping of physical button numbers with their functions below each button.

We assign a few variables including `CC0_value` through `CC5_value` to work with our potentiometers to send CC messages to the synth. `CC0_bankPin0` through `CC0_bankPin3` and `CC_bank0_value` through `CC_bank3_value` will be used to identify the position of the selector switch and used these values to choose which bank of parameters the knobs control at a give time.

### Synthesizer Parameters

Here is the full list of parameters on the DSP-G1 that can be adjusted with CC messages, as well as the default values we'll use upon startup:

```javascript
MIDI.sendControlChange( 7, 119, 1);   //volume
MIDI.sendControlChange( 1, 24, 1);    //LFO mod 24
MIDI.sendControlChange(16,  6, 1);    //LFO rate 12
MIDI.sendControlChange(20,  0, 1);    //LFO waveform 0-63 sine, 64-127 S/H
MIDI.sendControlChange(74, 80, 1);    //DC Filter cutoff
MIDI.sendControlChange(71, 70, 1);    //DC Filter resonance
MIDI.sendControlChange(82, 32, 1);    //DC Filter envelope Attack
MIDI.sendControlChange(83, 38, 1);    //DC Filter envelope Decay
MIDI.sendControlChange(28, 64, 1);    //DC Filter envelope Sustain
MIDI.sendControlChange(29, 32, 1);    //DC Filter envelope Release
MIDI.sendControlChange(81, 57, 1);    //DC Filter envelope modulation
MIDI.sendControlChange(76, 100, 1);   //DC Oscillator waveform* 100
MIDI.sendControlChange( 4,  0, 1);    //DC Oscillator wrap
MIDI.sendControlChange(21,  0, 1);    //DC Oscillator range
```
MIDI.sendControlChange(93, 0, 1); //DC Oscillator detune
MIDI.sendControlChange(73, 0, 1); //DC Oscillator envelope Attack
MIDI.sendControlChange(75, 12, 1); //DC Oscillator envelope Decay
MIDI.sendControlChange(31, 60, 1); //DC Oscillator envelope Sustain
MIDI.sendControlChange(72, 80, 1); //DC Oscillator envelope Release
/*Wavforms: 0 tri, 25 squarish, 50 pulse, 75 other squarish, 100 saw */

Loop

The main loop of the code does these things:

- Check position of the selector switch
- Check the bottom right Trellis pad to see if we are in momentary or held (latching) mode when the MIDI notes are played on the first three rows of eight pads
- Send MIDI note messages to the synth chip
- When potentiometer values change, send CC parameter value messages to the synth chip

Knob Readings

At the bottom of the code is a procedure called read_CC_Knobs() that is where a few special things happen!

To avoid constantly spamming the DSP-G1 with MIDI CC messages when nothing is changing, the procedure checks to see if the current value of each knob is different that the last time it changed. If not, don't send a message!

When a knob is changed, it's value is compared to the last value that knob had when the current bank was previously selected. To avoid having values suddenly jump, the messages are only sent when the pot value reaches the last value within a small tolerance.

Code Download

Here is the full code. Copy it, create a new sketch, paste in the code, save it, and then upload to your Feather. When the board restarts your synthesizer is ready to be played!

```cpp
#include <MIDI.h>
```

// SPDX-FileCopyrightText: 2018 John Edgar Park for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <MIDI.h>

©Adafruit Industries
#include "Adafruit_Trellis.h"
//use Trellis 0-11 and 16-27 as input for notes.
//use Trellis 12-15 and 28-31 for input to modes, sequencer, patches
//six potentiometers as CC value input
//rotary switch to choose banks of CC numbers for knobs

//John Park for Adafruit Industries

MIDI_CREATE_DEFAULT_INSTANCE();
Adafruit_Trellis matrix0 = Adafruit_Trellis(); //left Trellis
Adafruit_Trellis matrix1 = Adafruit_Trellis(); //right Trellis
Adafruit_TrellisSet trellis = Adafruit_TrellisSet(&matrix0, &matrix1);
#define NUMTRELLIS 2
#define numKeys (NUMTRELLIS * 16)
//define INTPIN A2
#define MOMENTARY 0
#define LATCHING 1
int holdMode = 0;
int lastHoldMode = 0;
static const unsigned LED = 13; // LED pin on the Feather

/*******************/
pick scale mode here:
//0=chromatic, 1= major, 2= minor, 3= dorian, 4= mixolydian, 5= harmonic minor
int scaleMode = 0; //Dorian
/*******************/
int octave = 1;
int transpose[6] = {0, 12, 24, 36, 48, 60}; //multiply notes depending on octave
//Trellis assignments

//First three rows map to MIDI Notes, bottom row is special functions

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
</table>
M8 | M9 | M10 | M11 | M12 | M13 | M14 | M15 |

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
</tr>
</thead>
</table>
M0 | M1 | M2 | M3 | M4 | M5 | M6 | M7 |

//Map physical Trellis grids to single logical grid
int logicalPads[32] = { 0, 1, 2, 3, 8, 9, 10, 11, 16, 17, 18, 19, 24, 25, 26, 27, 4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31 };

int padNoteMap[24] = { 16, 17, 18, 19, 20, 21, 22, 23, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7};

int padFuncMap[8] = { //note, only hold mode is implemented currently
0, // sequencer
1, // patch
2, // pause/play / recall
3, // record / store
4, // sequence I / patch I
5, // seq. II / patch II
6, // seq. III / patch III
7 //hold mode: LATCHING or MOMENTARY
};
// Define Scales
int scalesMatrix[6][24] = {
  // Chromatic
  { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23 },
  // Major
  { 0, 2, 4, 5, 7, 9, 11, 12, 12, 14, 16, 17, 19, 21, 23, 24, 24, 26, 28, 29, 31, 33, 35, 36 },
  // Minor
  { 0, 2, 3, 5, 7, 8, 10, 12, 12, 14, 15, 17, 19, 20, 22, 24, 24, 26, 27, 29, 31, 32, 34, 36 },
  // Dorian
  { 0, 2, 3, 5, 6, 9, 10, 12, 12, 14, 15, 17, 18, 21, 22, 24, 24, 26, 27, 29, 30, 33, 34, 36 },
  // Mixolydian
  { 0, 2, 4, 5, 7, 9, 10, 12, 12, 14, 16, 17, 19, 21, 22, 24, 24, 26, 28, 29, 31, 33, 34, 36 },
  // Harmonic Minor
  { 0, 2, 3, 5, 7, 8, 11, 12, 12, 14, 15, 17, 19, 20, 23, 24, 24, 26, 27, 29, 31, 32, 35, 36 }
};

// Set up selector switch pins to read digital input pins that set the CC bank
const int CC_bankPin0 = 9;
const int CC_bankPin1 = 10;
const int CC_bankPin2 = 11;
const int CC_bankPin3 = 12;

int CC_bankState0 = 0;
int CC_bankState1 = 0;
int CC_bankState2 = 0;
int CC_bankState3 = 0;

// current bank per knob. assume not on 0-3

// last bank per knob. assume not on 0-3
int lastBank[6] = {5, 5, 5, 5, 5, 5};

// Set up potentiometer pins
const int CC0_pin = A0; // potentiometer pin
const int CC1_pin = A1;
const int CC2_pin = A2;
const int CC3_pin = A3;
const int CC4_pin = A4;
const int CC5_pin = A5;

int CC_read[4];

// Store previous values to enable hysteresis and pick-up mode for knobs
int CC_bank0_lastValue[6] = {32, 32, 32, 32, 32, 32};
int CC_bank1_lastValue[6] = {32, 32, 32, 32, 32, 32};
```c
int CC_bank3_lastValue[6] = {32, 32, 32, 32, 32, 32};
int CC_bank0_value[6] = {0, 0, 0, 0, 0, 0};
int CC_bank1_value[6] = {0, 0, 0, 0, 0, 0};
int CC_bank2_value[6] = {0, 0, 0, 0, 0, 0};
int CC_bank3_value[6] = {0, 0, 0, 0, 0, 0};

void setup() {
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
  digitalWrite(LED, HIGH); // turn this on as a system power indicator
  pinMode(CC_bankPin0, INPUT_PULLUP); // loop this
  pinMode(CC_bankPin1, INPUT_PULLUP);
  pinMode(CC_bankPin2, INPUT_PULLUP);
  pinMode(CC_bankPin3, INPUT_PULLUP);

  // Default Arduino I2C speed is 100 KHz, but the HT16K33 supports
  // 400 KHz. We can force this for faster read & refresh, but may
  // break compatibility with other I2C devices...so be prepared to
  // comment this out, or save & restore value as needed.
  #ifdef ARDUINO_ARCH_SAMD
    Wire.setClock(400000L);
  #endif
  #ifdef __AVR__
    TWBR = 12; // 400 KHz I2C on 16 MHz AVR
  #endif

  trellis.begin(0x70, 0x71);
  for (uint8_t i=0; i<numKeys; i++) {
    trellis.setLED(i);
    trellis.writeDisplay();
    delay(15);
  }
  // then turn them off
  for (uint8_t i=0; i<numKeys; i++) {
    trellis.clrLED(i);
    trellis.writeDisplay();
    delay(15);
  }

  MIDI.begin(1);

  // All notes off MIDI panic if reset
  for(int n=0; n<128; n++) {
    MIDI.sendNoteOff(n, 127, 1);  // DC Filter cutoff
  }

  ///////////// DSP-G1 CC Parameter Settings Defaults (to be changed with knobs)
  MIDI.sendControlChange(7, 119, 1);  // volume
  MIDI.sendControlChange(1, 24, 1);  // LFO mod 24
  MIDI.sendControlChange(16, 6, 1);  // LFO rate 12
  MIDI.sendControlChange(20, 0, 1);  // LFO waveform 0-63 sine, 64-127 S/H
  MIDI.sendControlChange(74, 80, 1);  // DC Filter cutoff
  MIDI.sendControlChange(71, 70, 1);  // DC Filter resonance
  MIDI.sendControlChange(82, 32, 1);  // DC Filter envelope Attack
  MIDI.sendControlChange(83, 38, 1);  // DC Filter envelope Decay
  MIDI.sendControlChange(28, 64, 1);  // DC Filter envelope Sustain
  MIDI.sendControlChange(29, 32, 1);  // DC Filter envelope Release
  MIDI.sendControlChange(81, 57, 1);  // DC Filter envelope modulation
  MIDI.sendControlChange(76, 100, 1);  // DC Oscillator waveform* 100
  MIDI.sendControlChange(4, 0, 1);  // DC Oscillator wrap
  MIDI.sendControlChange(21, 0, 1);  // DC Oscillator range
  MIDI.sendControlChange(93, 0, 1);  // DC Oscillator detune
  MIDI.sendControlChange(73, 0, 1);  // DC Oscillator envelope Attack
  MIDI.sendControlChange(75, 12, 1);  // DC Oscillator envelope Decay
  MIDI.sendControlChange(31, 60, 1);  // DC Oscillator envelope Sustain
  MIDI.sendControlChange(72, 80, 1);  // DC Oscillator envelope Release

  /* Wavforms: 0 tri, 25 squarish, 50 pulse, 75 other squarish, 100 saw */
```
void loop(){
    //read selector switch pins to find CC knob bank
    CC_bankState0 = digitalRead(CC_bankPin0);//loop this
    CC_bankState1 = digitalRead(CC_bankPin1);
    CC_bankState2 = digitalRead(CC_bankPin2);
    CC_bankState3 = digitalRead(CC_bankPin3);
    int b; //to increment through current bank state values
    if (CC_bankState0 == LOW){ //low means it's the selected bank
        for (b=0;b<6;b++) {currentBank[b] = 0;}
        read_CC_Knobs(0);
    }
    else if (CC_bankState1 == LOW){
        for (b=0;b<6;b++) {currentBank[b] = 1;}
        read_CC_Knobs(1);
    }
    else if (CC_bankState2 == LOW){
        for (b=0;b<6;b++) {currentBank[b] = 2;}
        read_CC_Knobs(2);
    }
    else if (CC_bankState3 == LOW){
        for (b=0;b<6;b++) {currentBank[b] = 3;}
        read_CC_Knobs(3);
    }
    else { //4-7 have been picked, not connected
        for (b=0;b<6;b++) {currentBank[b] = 4;}
    }
    //Trellis MIDI out Keyboard
    delay(30); // 30ms delay is required, dont remove me!
    if (holdMode == MOMENTARY) {
        if (trellis.readSwitches()) { // If a button was just pressed or released...
            for (uint8_t i=0; i<numKeys; i++) { // go through every button
                if (trellis.justPressed(i)) { // if it was pressed, map what it actually
                    //remap the physical button to logical button. 'p' is logical pad, i is
                    physical pad. these names stink.
                    int p = logicalPads[i];
                    // Serial.print("logical pad: "); Serial.println(p);
                    if(p<24){ //it's a MIDI note pad, play a note
                        int padNote = logicalPads[i];
                        MIDI.sendNoteOn((scalesMatrix[00][padNoteMap[padNote]] +
                            transpose[octave]), 127, 1);
                        //uncomment for debugging:
                        /*
                            Serial.print("v"); Serial.println(p); //print which button pressed
                            trellis.setLED(i); //light stuff up
                            Serial.print("OUT: "); //print MIDI out command
                            Serial.print("\tChannel: "); Serial.print(1);
                            Serial.print("\tNote: "); Serial.println(scalesMatrix[00][padNoteMap[padNote]] +
                                transpose[octave]), 127, 1);
                        */
                    }
                    else if (p==31){ // last button on bottom row swaps hold modes
                        holdMode = !holdMode;
                        lastHoldMode = holdMode;
                        //Serial.println("Hold mode switch to Latching.");
                        for (uint8_t t =0; t<numKeys; t++) { //clear note LEDs
                            trellis.clrLED(n);
                        }
                        trellis.writeDisplay();
                    }
                    trellis.setLED(i); //light stuff up
                    trellis.writeDisplay();
                    delay(30);  
                }
                else if (p==31){ // last button on bottom row swaps hold modes
                    holdMode = !holdMode;
                    lastHoldMode = holdMode;
                    //Serial.println("Hold mode switch to Latching.");
                    for (uint8_t t =0; t<numKeys; t++) { //clear note LEDs
                        trellis.clrLED(n);
                    }
                    trellis.writeDisplay();
                }
                trellis.setLED(i); //light stuff up
                trellis.writeDisplay();
                delay(30);
            }
            else{
                Serial.println("not a note");
            }
    }
    else if (p==31){ // last button on bottom row swaps hold modes
    holdMode = !holdMode;
    lastHoldMode = holdMode;
    //Serial.println("Hold mode switch to Latching.");
    for (uint8_t t =0; t<numKeys; t++) { //clear note LEDs
    trellis.clrLED(n);
    }
    trellis.writeDisplay();
    } 
    else {
        Serial.println("not a note");
}
if (trellis.justReleased(i)) { // if it was released, turn it off
    int p = logicalPads[i];
    //Serial.print("logical pad: "); Serial.println(p);
    if(p<24) { // it's a MIDI note pad, play a note
        int padNote = logicalPads[i];
        MIDI.sendNoteOff((scalesMatrix[scaleMode][padNoteMap[padNote]] +
        transpose[octave]), 127, 1);
        //Serial.print("^"; Serial.println(p);
        for (uint8_t i=0; i<numKeys; i++) {
            trellis.clrLED(i);
            trellis.writeDisplay();
        }
        trellis.setLED(i); // keep last one that was pressed turned on
        // uncomment for debugging
        /*
        Serial.print("OUT: ");
        Serial.print("Channel: "); Serial.print(1);
        Serial.print("Note: "); Serial.print(scalesMatrix[scaleMode]
        [padNoteMap[padNote]]+ transpose[octave]);
        */
        //Serial.print("\tValue: "); Serial.println("0");
    }
    else if (p==31) { // last button on bottom row swaps hold modes
        holdMode = !holdMode;
        lastHoldMode = holdMode;
        for(int n=0; n<128; n++){
            MIDI.sendNoteOff(n, 127, 1); //DC Filter cutoff
        }
        for (uint8_t j=0; j<24; j++) { // clear all note pads
            trellis.clrLED(j);
        }
        trellis.clrLED(i); // clear holdMode pad
        trellis.writeDisplay();
        //Serial.println("holdMode switch to Momentary.");
    } else{ // not a note
    }
}
}
if (holdMode == LATCHING) {
    if (trellis.readSwitches()) {
        for (uint8_t i=0; i<numKeys; i++) {
            if (trellis.justPressed(i)) {
                int p = logicalPads[i];
                //Serial.print("logical pad: "); Serial.println(p);
                if(p<24) {
                    int padNote = logicalPads[i];
                    //Serial.print(" pad note: "); Serial.println(padNote);
                    //Serial.print("v"); Serial.println(i);
                    if (trellis.isLED(i)){ // Alternate the button
                        MIDI.sendNoteOff((scalesMatrix[scaleMode][padNoteMap[padNote]] +
                        transpose[octave]), 127, 1);
                        trellis.clrLED(i);
                    }
                    else{
                        MIDI.sendNoteOn((scalesMatrix[scaleMode][padNoteMap[padNote]] +
                        transpose[octave]), 127, 1);
                        trellis.setLED(i);
                    }
                }
                else if (i==31){ // last button on bottom row swaps hold modes
                    holdMode = !holdMode;
                    lastHoldMode = holdMode;
                    for(int n=0; n<128; n++){
                        MIDI.sendNoteOff(n, 127, 1); //DC Filter cutoff
                    }
                    for (uint8_t j=0; j<24; j++) { // clear all note pads
                        trellis.clrLED(j);
                    }
                    trellis.clrLED(i); // clear holdMode pad
                    trellis.writeDisplay();
                    //Serial.println("holdMode switch to Momentary.");
                } else{ // not a note
    }
}
}
delay(30);
}
else{
  Serial.println("not a note");
}
trellis.writeDisplay();
}
}
//clear key LEDs after a hold mode change
if(holdMode != lastHoldMode){
  //Serial.print("Hold mode: "); Serial.println(holdMode);
  //Serial.print("Last hold mode: "); Serial.println(lastHoldMode);
  for (uint8_t j=0; j<24; j++) { //clear all note pads
    trellis.clrLED(j);
  }
  trellis.writeDisplay();
}

void read_CC_Knobs(int CC_bank){
  int AnalogPin[6] = {A0, A1, A2, A3, A4, A5}; //define the analog pins
  int CCKnob[6] = {0, 1, 2, 3, 4, 5}; //define knobs
  // choose which CC numbers are on each knob per bank
  // definitions are at bottom of code
  //Note: can set knobs to "never change" by repeating CC numbers per array
  int CCNumber[6]; //variable array to store the CC numbers
  if(CC_bank == 0){ //set the CC numbers per knob per bank
    //OSC
    int CCNumberChoice[6] = {71, //Filter resonance bottom of left knobs
                             74, //Filter cutoff top of left two knobs
                             21, //DCO range upper row
                             93, //DCO detune
                             4,  //DCO wrap
                             76 //DCO waveform
    };
    for(int n=0; n<6; n++){
      CCNumber[n] = CCNumberChoice[n];
    }
  }
  else if(CC_bank == 1){
    //Envelope (for main oscillators)
    int CCNumberChoice[6] = {71, //Filter resonance
                             74, //Filter cutoff top of left two knobs
                             82, //VCA Attack
                             83, //VCA Decay
                             28, //VCA Sustain
                             29  //VCA Release
    };
    for(int n=0; n<6; n++){
      CCNumber[n] = CCNumberChoice[n];
    }
  }
  else if(CC_bank == 2){
    //LFO low frequency oscillator for main OSC
    int CCNumberChoice[6] = {71, //Filter resonance
                             74, //Filter cutoff top of left two knobs
                             1,  //LFO mod
                             16, //LFO rate
                             20, //LFO waveform shape
                             81 //Filter mod
    };
    for(int n=0; n<6; n++){
      CCNumber[n] = CCNumberChoice[n];
    }
  }
}
else if(CC_bank == 3){
    //Envelope (for filter)
    int CCNumberChoice[6] = {71, //Filter resonance
                            74, //Filter cutoff top of left two knobs
                               1, //Filter A
                              16, //Filter D
                             20, //Filter S
                             81 //Filter R
                          };
    for(int n=0; n<6; n++){
        CCNumber[n] = CCNumberChoice[n];
    }
}

//Knob pick-up mode -- to deal with multiple knob banks,
//value doesn't get sent until knob reaches previous ("last") position
//NOTE: ugly way to do this, clean it up so a matrix is used instead
//per bank loop currently

//Send CC values only when they change beyond last value to avoid getting stuck
//between two values
//thanks to Groovesizer Foxtrot code for this idea: https://groovesizer.com/
//thanks to Todbot for the delta hysteresis idea

if(CC_bank == 0){ //first bank is selected
    for(int k=0; k<6; k++)//loop through each of six pots
    if(currentBank[k] != lastBank[k]){ //if the current bank for current knob
        //digitalWrite(LED, LOW);
        //is different than the last bank for the current knob:
        //read CC values from potentiometers
        CC_read[CC_bank] = analogRead(AnalogPin[k]);
        CC_bank0_value[k] = map(CC_read[CC_bank], 0, 1023, 0, 127); // remap range
to 0-127
        //check against last read when we were on this bank, only send CC if value
        //of knob is withing a certain delta of the last value
        if((abs(CC_bank0_lastValue[k] - CC_bank0_value[k])) < 3){ //hysteresis for
            Serial.println("PICKUP ACHIEVED");
            //digitalWrite(LED, HIGH);
            Serial.print("\nCC_bank0 value for knob: "); Serial.print(k);
            Serial.print("\t"); Serial.println(CC_bank0_value[k]);
            Serial.print("CC_bank0 lastValue for knob: "); Serial.print(k);
            Serial.print("\t"); Serial.println(CC_bank0_lastValue[k]);
            lastBank[k] = 0;
        }
    }
    else if(currentBank[k] == lastBank[k]){
        //read CC values from potentiometers
        CC_read[CC_bank] = analogRead(AnalogPin[k]);
        CC_bank0_value[k] = map(CC_read[CC_bank], 0, 1023, 0, 127); // remap range
to 0-127
        //check against last read, only send CC if value has changed since last
        read by a certain delta
        if((abs(CC_bank0_lastValue[k] - CC_bank0_value[k])) > 3){ //hysteresis for
            Serial.println("PICKUP ACHIEVED");
            //digitalWrite(LED, HIGH);
            Serial.print("\nCC_bank0 value for knob: "); Serial.print(k);
            Serial.print("\t"); Serial.println(CC_bank0_value[k]);
            Serial.print("CC_bank0 lastValue for knob: "); Serial.print(k);
            Serial.print("\t"); Serial.println(CC_bank0_lastValue[k]);
            MIDI.sendControlChange(CCNumber[k], CC_bank0_value[k], 1);
            Serial.print("CC number: "); Serial.println(CCNumber[k]);
            CC_bank0_lastValue[k] = CC_bank0_value[k];
            lastBank[k] = 0;
        }
    }
}

if(CC_bank == 1){ //second bank is selected
for(int k=0; k<6; k++)
{
  //loop through each of six pots
  if(currentBank[k] != lastBank[k]){
    //read CC values from potentiometers
    CC_read[CC_bank] = analogRead(AnalogPin[k]);
    CC_bank1_value[k] = map(CC_read[CC_bank], 0, 1023, 0, 127); // remap range to 0-127
    //check against last read when we were on this bank, only send CC if value
    //of knob is within a certain delta of the last value
    if((abs(CC_bank1_lastValue[k] - CC_bank1_value[k])) < 3){ //hysteresis for spinning knob too fast
      Serial.println("Pickup achieved.");
      Serial.print("\nCC bank1 value for knob: "); Serial.print(k);
      Serial.print("\nCC_bank1_value for knob: "); Serial.print(k);
      Serial.print("\nCC_bank1_lastValue for knob: "); Serial.print(k);
      MIDI.sendControlChange(CCNumber[k], CC_bank1_value[k], 1);
      //Serial.print("CC number: "); Serial.println(CCNumber[k]);
      CC_bank1_lastValue[k] = CC_bank1_value[k];
      lastBank[k] = 1;
    }
  }
  else if(currentBank[k] == lastBank[k]){
    //read CC values from potentiometers
    CC_read[CC_bank] = analogRead(AnalogPin[k]);
    CC_bank1_value[k] = map(CC_read[CC_bank], 0, 1023, 0, 127); // remap range to 0-127
    //check against last read, only send CC if value has changed since last
    //read by a certain delta
    if((abs(CC_bank1_lastValue[k] - CC_bank1_value[k])) > 3){ //hysteresis for spinning knob too fast
      Serial.print("\nCC_bank1 value for knob: "); Serial.print(k);
      Serial.print("\nCC_bank1 value for knob: "); Serial.print(k);
      Serial.print("\nCC_bank1 lastValue for knob: "); Serial.print(k);
      MIDI.sendControlChange(CCNumber[k], CC_bank1_value[k], 1);
      //Serial.print("CC number: "); Serial.println(CCNumber[k]);
      CC_bank1_lastValue[k] = CC_bank1_value[k];
      lastBank[k] = 1;
    }
  }
}

if(CC_bank == 2){ //third bank is selected
  for(int k=0; k<6; k++)
  {
    //loop through each of six pots
    if(currentBank[k] != lastBank[k]){
      //read CC values from potentiometers
      CC_read[CC_bank] = analogRead(AnalogPin[k]);
      CC_bank2_value[k] = map(CC_read[CC_bank], 0, 1023, 0, 127); // remap range to 0-127
      //check against last read when we were on this bank, only send CC if value
      //of knob is within a certain delta of the last value
      if((abs(CC_bank2_lastValue[k] - CC_bank2_value[k])) < 3){ //hysteresis for spinning knob too fast
        Serial.println("Pickup achieved.");
        Serial.print("\nCC_bank2 value for knob: "); Serial.print(k);
        Serial.print("\nCC_bank2 value for knob: "); Serial.print(k);
        Serial.print("\nCC_bank2 lastValue for knob: "); Serial.print(k);
        MIDI.sendControlChange(CCNumber[k], CC_bank2_value[k], 1);
        //Serial.print("CC number: "); Serial.println(CCNumber[k]);
        CC_bank2_lastValue[k] = CC_bank2_value[k];
        lastBank[k] = 2;
      }
    }
    else if(currentBank[k] == lastBank[k]){
      //read CC values from potentiometers
      CC_read[CC_bank] = analogRead(AnalogPin[k]);
      CC_bank2_value[k] = map(CC_read[CC_bank], 0, 1023, 0, 127); // remap range to 0-127
      //check against last read, only send CC if value has changed since last
      //read by a certain delta
      if((abs(CC_bank2_lastValue[k] - CC_bank2_value[k])) > 3){ //hysteresis for spinning knob too fast
        Serial.print("\nCC_bank2 value for knob: "); Serial.print(k);
        Serial.print("\nCC_bank2 value for knob: "); Serial.print(k);
        Serial.print("\nCC_bank2 lastValue for knob: "); Serial.print(k);
      }
    }
  }
}
Serial.print("CC_bank2_lastValue for knob: "); Serial.print(k);
Serial.print(
"	"); Serial.println(CC_bank2_lastValue[k]);
MIDI.sendControlChange(CCNumber[k], CC_bank2_value[k], 1);
//Serial.print("CC number: "); Serial.println(CCNumber[k]);
CC_bank2_lastValue[k] = CC_bank2_value[k];
lastBank[k] = 2;
}
}
}
}
if(CC_bank == 3){ //fourth bank is selected
for(int k=0; k<6; k++) { //loop through each of six pots
if(currentBank[k] != lastBank[k]){
    //read CC values from potentiometers
    CC_read[CC_bank] = analogRead(AnalogPin[k]);
    CC_bank3_value[k] = map(CC_read[CC_bank], 0, 1023, 0, 127); // remap range
to 0-127
    //check against last read when we were on this bank, only send CC if value
    //of knob is within a certain delta of the last value
    if((abs(CC_bank3_lastValue[k] - CC_bank3_value[k])) < 3){ //hysteresis for
    Serial.println("Pickup achieved.");
    Serial.print("\nCC_bank3 value for knob: "); Serial.print(k);
    Serial.print("\t"); Serial.println(CC_bank3_value[k]);
    Serial.print("CC_bank3 lastValue for knob: "); Serial.print(k);
    Serial.print("\t"); Serial.println(CC_bank3_lastValue[k]);
lastBank[k] = 3;
}
else if(currentBank[k] == lastBank[k]){ //read CC values from potentiometers
    CC_read[CC_bank] = analogRead(AnalogPin[k]);
    CC_bank3_value[k] = map(CC_read[CC_bank], 0, 1023, 0, 127); // remap
    range to 0-127
    //check against last read, only send CC if value has changed since last
    read by a certain delta
    if((abs(CC_bank3_lastValue[k] - CC_bank3_value[k])) > 3){ //hysteresis for
    Serial.print("\nCC_bank3 value for knob: "); Serial.print(k);
    Serial.print("\t"); Serial.println(CC_bank3_value[k]);
    Serial.print("CC_bank3 lastValue for knob: "); Serial.print(k);
    Serial.print("\t"); Serial.println(CC_bank3_lastValue[k]);
    MIDI.sendControlChange(CCNumber[k], CC_bank3_value[k], 1);
    //Serial.print("CC number: "); Serial.println(CCNumber[k]);
    CC_bank3_lastValue[k] = CC_bank3_value[k];
lastBank[k] = 3;
}
} else { //send CC, but only if value has changed since last read
    MIDI.sendControlChange(CCNumber[k], CC_bank3_value[k], 1);
    //Serial.print("CC number: "); Serial.println(CCNumber[k]);
    CC_bank3_lastValue[k] = CC_bank3_value[k];
lastBank[k] = 3;
}
Play Music

Time to play some music and make awesome, synth-y sounds! Plug in an 1/8” (3.5mm) phono cable to the back of the synth and run it into your loudest amp! (Powered speakers or a pair headphones will work, too.)

You can download and print this graphic file to overlay on your synth to help you with the control layout. Note, not all features on the graphic have been implemented yet. But you can see where there's room for future growth with sequencing, patch store and recall and more! Maybe you'll make an arpeggiator function instead. It's up to you!

[Synth diagram]

synthGraphics01.pdf