Trellis M4 Synth Design Tool
Created by John Park

https://learn.adafruit.com/synthesizer-design-tool

Last updated on 2022-01-28 11:52:39 AM EST
# Table of Contents

## Overview

## Synthesis Concepts
- Modules and Patches 7
- Oscillators 7
- Sine Wave 8
- Triangle Wave 8
- Sawtooth Wave 9
- Square Wave 10
- Audio Output 11
- Mixers 12
- Envelopes 14
- Effects 15

## Audio System Design Tool
- Audio Library 16
- Audio System Design Tool 16
- GUI Tour 17
- Build a Patch 18
- Info 20
- Code Export and Import 21

## Arduino IDE Setup

## Using with Arduino IDE
- Install SAMD Support 24
- Install Adafruit SAMD 25
- Install Drivers (Windows 7 & 8 Only) 26
- Blink 28
- Successful Upload 29
- Compilation Issues 30
- Manually bootloading 31
- Ubuntu & Linux Issue Fix 31

## Arduino Libraries
- Install Libraries 32
- Adafruit NeoPixel 32
- Adafruit DMA NeoPixel 32
- Adafruit Unified Sensor 32
- Adafruit SPIFlash 33
- Adafruit Keypad 33
- MIDI USB 33
- ADXL343 33
- NeoTrellis M4 34
- SdFat - Adafruit Fork 34
- Audio - Adafruit Fork 34

## Audio Library Examples on Trellis M4
- Tonesweep 36
- Bootloader Mode 39
- Waveform Pitch 40
- Waveform 43
• Waveform to Audio Output

Full Synth Demo

• Four Oscillator Synthesizer
• Arduino Code
Overview

You can build your own synthesizer using the NeoTrellis M4 with the PJRC Audio System Design Tool and Audio library for Arduino! In this guide you'll learn how to patch together waveform oscillators, filters, envelopes, effects, and mixers. Then, you'll learn how to control notes and parameters with the Trellis M4’s buttons and accelerometer.

Soon you'll be playing ripping synth leads, fat bass lines, and lush pads with a synthesizer of your own design!

Adafruit NeoTrellis M4 with Enclosure and Buttons Kit Pack
So you've got a cool/witty name for your band, a Soundcloud account, a 3D-printed Daft Punk...
https://www.adafruit.com/product/4020

Cell-phone TRRS Headset - Earbud Headphones w/ Microphone
These earbud headphones are the perfect accessory for your FONA - they've been tested to work with our modules - but can be used with any iOS or Android device that uses a TRRS...
https://www.adafruit.com/product/1966
Monoprice 5-Watt Guitar Amplifier, Portable Recorder
Always be ready rock out or record with this 5-watt Guitar Amplifier, Portable Recorder, and USB Audio Interface from Monoprice. This handheld 5-watt amp...
https://www.adafruit.com/product/4402

Stereo 3.5mm Plug/Plug Audio Cable - 6 feet
This basic cable comes with two 3.5mm (1/8" headphone jack size) stereo connectors. It's fairly straight forward, you'll commonly need these to connect two audio devices...
https://www.adafruit.com/product/876

USB Powered Speakers
Add some extra boom to your audio project with these powered loudspeakers. We sampled half a dozen different models to find ones with a good frequency response, so you'll get...
https://www.adafruit.com/product/1363

Synthesis Concepts
Before we design and code our synths, let's define a few fundamental elements of music synthesis.
Modules and Patches

Early synthesizers from Robert Moog and Don Buchla tended to be made up of discreet modules connected via patch cables. So, the flowchart-like path of connected nodes used to wire up the components of a synthesizer are often called patches.

Oscillators

The most fundamental element of a synthesizer is the oscillator. Its job is to create sound by producing a signal that varies in strength quickly over time, and usually with a repetitive, periodic cycle. This is typically an oscillation of AC voltage moving from, say, -1V to +1V.

How quickly does this wave oscillate? Anywhere from 20Hz to 20,000Hz is the typical range for human hearing. The frequency of the oscillator determines the pitch of the sound.

The shape of the oscillator's signal is also important. A smooth sine wave will have a very pure sounding tone consisting only of the fundamental frequency, while a more jagged sawtooth will be rich with additional harmonics, lending a "buzzy" sound on top of the fundamental.

Here are some typical waveforms used for audio rate oscillators (the images were made using VCV Rack (https://adafruit.it/C-b) an Open Source Eurorack modular synthesizer program).
Sine Wave

Triangle Wave
Square Wave

One other notable sound source worth mentioning is Noise. It can be used for adding texture and percussive qualities to synthesized waveforms, as well as for more exotic applications, such as wind and wave crash sounds.
Audio Output

One of the most basic patches we can make is to wire an oscillator to an audio output node. In the Audio System Design Tool (which we'll cover more closely later in this guide), it will look something like this:
Mixers

A mixer is used to combine the signals of multiple audio sources. Each signal is fed into a separate channel -- four-channel mixers are typical. Each channel has an attenuator (usually a fader or knob in the real world) used to adjust how much of that channel's signal is combined in the mix. The mix output can then be patched to the audio output node, or other modules in a more complex patch.

Here's an example of what happens when we mix a sine waveform with a white noise source.
Envelopes

Rather than always playing constant tones with not starts or stops, we’ll often want to activate the sound only at specific times, such as when a button is pressed. This is accomplished by adjusting the waveform’s signal strength with an amplifier.

In its simplest form, this can sound like a harsh ON/OFF with the tone abruptly starting and stopping. Usually we prefer the sound of a more nuanced loudness envelope so that the sound seems to have been plucked, or struck, or blown as with a stringed instrument, woodwind, piano, percussion, and so on. The envelope that shapes the amplifier is described in stages: Attack, Decay, Sustain, and Release, often abbreviated as ADSR.

![ADS envelope diagram]

With this ADSR envelope we can specify the following parameter:

- **Attack**: amount of time it takes to amplify from zero to full volume
- **Decay**: amount of time it takes to drop from full volume to the sustain volume
- **Sustain**: volume level to maintain until the note is released
- **Release**: amount of time to return from the sustain volume level to zero

Some envelope models are even more nuanced and add parameters for Delay (time before the attack begins), and Hold (time between attack and decay).
Effects

Once we know how to run oscillators through envelopes and into a mixer, it’s time to send some of the signals through effects! Typical effects available on a synthesizer include:

- Delay
- Reverb
- Chorus
- Flanger
- Bitcrusher

These can be placed nearly anywhere in the signal path of the synthesizer. We’ll create a patch later that runs individual oscillator waveforms through their own effects on their way to the mixers. This diagram shows one such patch, which we’ll use later to create the modules and patches for our Arduino code.
Audio System Design Tool

Audio Library

The key element to creating our synthesizer for the Trellis M4 is the Audio library by PJRC (https://adafru.it/Dfv) for Arduino. This tremendous library is written by Paul Stoffregen for the Teensy line of microcontrollers, and has been the backbone of many, many amazing audio projects on Teensy. It has now been ported to run on NeoTrellis M4, opening up a whole new world of audio projects that can use the RGB lighted button grid and accelerometer for control!

While you can write your Audio library Arduino patches entirely in code, the secret weapon for crafting your synth is the Node-RED based GUI, the Audio System Design Tool.

The Microcontroller Audio Workshop run by Paul Stoffregen at HaD Superconference 2015 with videos and extensive documentation are an invaluable resource for learning more about the Audio library. Check it out: https://hackaday.io/project/8292-microcontroller-audio-workshop-had-supercon-2015

Audio System Design Tool

The Audio System Design Tool is a graphical interface for creating modules and patches for the Audio library written by PJRC (https://adafru.it/Dfv).
From the tool's readme:

The Audio System Design Tool lets you easily draw a system to process 16 bit, 44.1 kHz streaming audio while your Arduino sketch also runs.

Export will generate code to copy into the Arduino editor, to implement your system.

Most objects provide simple functions you can call from setup() or loop() to control your audio project!

It runs in a web browser, so you can simple open it up by clicking this link (it will open into a new browser window, so if it won't launch, you may need to adjust your browser's pop-up window settings to allow it):

Audio System Design Tool for Teensy Audio Library
https://adafru.it/Dkn

Let's take a look at the interface and build our first patch.

GUI Tour

The GUI is made up of three sections:

- Nodes panel -- where all of the nodes (also called "objects" or "modules") are that can be used, grouped by types. Use the mouse wheel to scroll through this long list!
- Canvas -- where you'll drag nodes and connect them to build your patches
- Info panel -- helpful information on your currently selected node is displayed in this panel
Build a Patch

In order to build a patch, you will drag objects from the Nodes panel onto the canvas, and then patch the nodes by drag-connecting from an output port of one object to an input port on another.

Waveform

First, scroll down to the synth category and find the waveform node. Drag it to the canvas.
Audio Output

Next, scroll up to the output category and find the dacs node. This is the stereo Digital-to-Analog-Converter we use to output sound over the headphone jack on the NeoTrellis M4.

Patch Cables

Now, we'll patch the output port of the waveform node to the two input ports (stereo L and R) of the dacs node. This is done by simply clicking the mouse on one port, dragging, and releasing on the other port.

In many cases you'll want to connect the output port of one node to the input ports of multiple nodes. This is easy to do -- simply drag multiple patch cables
Here's an example of using mixers to allow different levels of a waveform and pink noise to be mixed into each side of the stereo dacs audio output.

Info

When any node is selected, the Info panel will provide useful information on the node, such as a summary, the port mapping, how the functions will be used in the Arduino sketch for that node, and a pointer to any example files in the Arduino library.
Code Export and Import

On the next page we'll look at setting up the Arduino IDE to use the Audio library and write sketches. But, before we move on, we'll look at one last important feature of the Audio System Design Tool -- code Export and Import.

In order to generate the code snippet that defines our nodes and patches, click the Export button. This will open the Export to Arduino window. Here, you can copy and paste the source code and then close the window.

You can also import code that you have adjusted in Arduino code, so if you happen to rewire things there but then want to continue working on the patch in the GUI, you'll use the Import button instead.

Before we dive into using the Audio System Design Tool generated code snippet, we'll get things set up for using Arduino and the Audio library on the NeoTrellis M4.

Arduino IDE Setup

The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using version 1.8 or higher for this guide.

Arduino IDE Download

https://adafru.it/f1P

©Adafruit Industries
After you have downloaded and installed the latest version of Arduino IDE, you will need to start the IDE and navigate to the Preferences menu. You can access it from the File menu in Windows or Linux, or the Arduino menu on OS X.

A dialog will pop up just like the one shown below.

We will be adding a URL to the new Additional Boards Manager URLs option. The list of URLs is comma separated, and you will only have to add each URL once. New Adafruit boards and updates to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

To find the most up to date list of URLs you can add, you can visit the list of third party board URLs on the Arduino IDE wiki (https://adafruit.it/f7U). We will only need to add one URL to the IDE in this example, but you can add multiple URLs by separating them with commas.

©Adafruit Industries
them with commas. Copy and paste the link below into the Additional Boards Manager URLs option in the Arduino IDE preferences.

https://adafruit.github.io/arduino-board-index/package_adafruit_index.json

Here's a short description of each of the Adafruit supplied packages that will be available in the Board Manager when you add the URL:

- Adafruit AVR Boards - Includes support for Flora, Gemma, Feather 32u4, ItsyBitsy 32u4, Trinket, & Trinket Pro.
- Adafruit SAMD Boards - Includes support for Feather M0 and M4, Metro M0 and M4, ItsyBitsy M0 and M4, Circuit Playground Express, Gemma M0 and Trinket M0
- Arduino Leonardo & Micro MIDI-USB - This adds MIDI over USB support for the Flora, Feather 32u4, Micro and Leonardo using the [arcore project](https://adafruit.it/eSI).

If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

Once done click OK to save the new preference settings. Next we will look at installing boards with the Board Manager.

Now continue to the next step to actually install the board support package!
Using with Arduino IDE

The Feather/Metro/Gemma/QTPy/Trinket M0 and M4 use an ATSAMD21 or ATSAMD51 chip, and you can pretty easily get it working with the Arduino IDE. Most libraries (including the popular ones like NeoPixels and display) will work with the M0 and M4, especially devices & sensors that use I2C or SPI.

Now that you have added the appropriate URLs to the Arduino IDE preferences in the previous page, you can open the Boards Manager by navigating to the Tools->Board menu.

Once the Board Manager opens, click on the category drop down menu on the top left hand side of the window and select All. You will then be able to select and install the boards supplied by the URLs added to the preferences.

Remember you need SETUP the Arduino IDE to support our board packages - see the previous page on how to add adafruit's URL to the preferences.

Install SAMD Support

First up, install the latest Arduino SAMD Boards (version 1.6.11 or later)

You can type Arduino SAMD in the top search bar, then when you see the entry, click Install
Install Adafruit SAMD

Next you can install the Adafruit SAMD package to add the board file definitions

Make sure you have Type All selected to the left of the Filter your search... box

You can type Adafruit SAMD in the top search bar, then when you see the entry, click Install

Even though in theory you don't need to - I recommend rebooting the IDE

Quit and reopen the Arduino IDE to ensure that all of the boards are properly installed. You should now be able to select and upload to the new boards listed in the Tools->Board menu.
Select the matching board, the current options are:

- Feather M0 (for use with any Feather M0 other than the Express)
- Feather M0 Express
- Metro M0 Express
- Circuit Playground Express
- Gemma M0
- Trinket M0
- QT Py M0
- ItsyBitsy M0
- Hallowing M0
- Crickit M0 (this is for direct programming of the Crickit, which is probably not what you want! For advanced hacking only)
- Metro M4 Express
- Grand Central M4 Express
- ItsyBitsy M4 Express
- Feather M4 Express
- Trellis M4 Express
- PyPortal M4
- PyPortal M4 Titano
- PyBadge M4 Express
- Metro M4 Airlift Lite
- PyGamer M4 Express
- MONSTER M4SK
- Hallowing M4
- MatrixPortal M4
- BLM Badge

Install Drivers (Windows 7 & 8 Only)

When you plug in the board, you'll need to possibly install a driver
Click below to download our Driver Installer

Download Latest Adafruit Drivers package
https://adafruit.it/mb8

Download and run the installer

Run the installer! Since we bundle the SiLabs and FTDI drivers as well, you'll need to click through the license

Select which drivers you want to install, the defaults will set you up with just about every Adafruit board!
Click Install to do the installin'

Blink

Now you can upload your first blink sketch!

Plug in the M0 or M4 board, and wait for it to be recognized by the OS (just takes a few seconds). It will create a serial/COM port, you can now select it from the drop-down, it'll even be 'indicated' as Trinket/Gemma/Metro/Feather/ItsyBitsy/Trellis!
Please note, the QT Py and Trellis M4 Express are two of our very few boards that does not have an onboard pin 13 LED so you can follow this section to practice uploading but you won't see an LED blink!

Now load up the Blink example

```c
void setup() {
    pinMode(13, OUTPUT);
}

void loop() {
    digitalWrite(13, HIGH);  // turn the LED on (HIGH is the voltage level)
    delay(1000);              // wait for a second
    digitalWrite(13, LOW);   // turn the LED off by making the voltage LOW
    delay(1000);             // wait for a second
}
```

And click upload! That's it, you will be able to see the LED blink rate change as you adapt the delay() calls.

If you are having issues, make sure you selected the matching Board in the menu that matches the hardware you have in your hand.

**Successful Upload**

If you have a successful upload, you'll get a bunch of red text that tells you that the device was found and it was programmed, verified & reset.
After uploading, you may see a message saying "Disk Not Ejected Properly" about the ...BOOT drive. You can ignore that message: it's an artifact of how the bootloader and uploading work.

**Compilation Issues**

If you get an alert that looks like

```
Cannot run program "[runtime.tools.arm-none-eabi-gcc.path]\bin\arm-none-eabi-g++"
```

Make sure you have installed the Arduino SAMD boards package, you need both Arduino & Adafruit SAMD board packages
Manually bootloading

If you ever get in a 'weird' spot with the bootloader, or you have uploaded code that crashes and doesn't auto-reboot into the bootloader, click the RST button twice (like a double-click) to get back into the bootloader.

The red LED will pulse and/or RGB LED will be green, so you know that its in bootloader mode.

Once it is in bootloader mode, you can select the newly created COM/Serial port and re-try uploading.

You may need to go back and reselect the 'normal' USB serial port next time you want to use the normal upload.

Ubuntu & Linux Issue Fix

Follow the steps for installing Adafruit's udev rules on this page. (https://adafruit.it/iOE)

Arduino Libraries

OK now that you have Arduino IDE set up, drivers installed if necessary and you've practiced uploading code, you can start installing all the Libraries we'll be using to program it.

There's a lot of libraries!
Install Libraries

Open up the library manager...

And install the following libraries:

**Adafruit NeoPixel**

This will let you light up the LEDs on the front

**Adafruit DMA NeoPixel**

This adds a special NeoPixel library that uses DMA so the NeoPixel stuff happens without processor time taken.

**Adafruit Unified Sensor**

The underlying sensor library for ADXL343 support
Adafruit SPIFlash

This will let you read/write to the onboard FLASH memory with super-fast QSPI support

Adafruit Keypad

Our Keypad support library (for reading the button matrix)

MIDI USB

So you can have the Trellis M4 act like a MIDI device over USB

ADXL343

The ADXL343 Library which provides accelerometer support
NeoTrellis M4

The NeoTrellis_M4 Library that handles MIDI, LEDs & button presses

SdFat - Adafruit Fork

Our fork of the SdFat library provides read/write access to FAT16/FAT32 file systems on SD/SDHC flash cards.

Audio - Adafruit Fork

Our fork of the Audio library provides a toolkit for building streaming audio projects.

Audio Library Examples on Trellis M4

Now that you've got the Arduino IDE setup for the Trellis M4 and the Audio library, we'll make a simple test of the audio system.

First open the Arduino IDE and create a new sketch by clicking File > New
Then, copy the code shown below and paste it into the new sketch.

```cpp
/*
   Audio Library on Trellis M4
   Demo of the audio sweep function.
   The user specifies the amplitude,  
   start and end frequencies (which can sweep up or down) 
   and the length of time of the sweep. 
*/

#include &lt;Audio.h&gt;
// Paste your Audio System Design Tool code below this line:

//  GUItool: end automatically generated code

float t_ampx = 0.05;  // Amplitude
int t_lox = 10;  // Low frequency
int t_hix = 22000;  // High frequency
float t_timex = 10; // Length of time of the sweep in seconds

void setup(void)
{
    Serial.begin(9600);
    //while (!Serial) ;
    delay(3000);
    AudioMemory(6);
    Serial.println("setup done");
    Serial.println("AudioSynthToneSweep - begin failed");
    while(1);
} // wait for the sweep to end
while(tonesweep1.isPlaying());
// and now reverse the sweep
if(!tonesweep1.play(t_ampx,t_hix,t_lox,t_timex)) {
    Serial.println("AudioSynthToneSweep - begin failed");
    while(1);
} // wait for the sweep to end
while(tonesweep1.isPlaying());
Serial.println("Done");

void loop(void)
{
}
```
Tonesweep

Create a tonesweep node and a dacs node.

Patch the output port of the tonesweep to both inputs of the dacs.

Double-click the dacs node to open the rename dialog box.

Rename the node "audioOutput" and then press the "OK" button (hitting Enter on the keyboard doesn't work for me).

Next, press the Export button to generate the source code snippet. Since the tool was designed for use with the Teensy, some of the library imports listed at the top of the code aren't necessary for use with the Trellis M4. So, only select and copy the section surrounded by the // GUItool: lines.
Next, head back to the Arduino IDE and paste the code into the section at the top where indicated.

Save this sketch as ToneSweep_TrellisM4.ino

This is what the final sketch looks like after adding the Audio GUI code:

```cpp
#include <Audio.h>

// GUItool: begin automatically generated code
AudioSynthToneSweep tonesweep1;  //xy=531.0833129882812,166.08334350585938
AudioOutput AnalogStereo audioOutput;  //xy=350.0462951660156,408.3796081542569
AudioConnection patchCord1(tonesweep1, 0, audioOutput, 0);
AudioConnection patchCord2(tonesweep1, 0, audioOutput, 1);
// GUItool: end automatically generated code

float t_ampx = 0.05;  // Amplitude
int t_lox = 10;  // Low frequency
int t_hix = 22000;  // High frequency
float t_timex = 10;  // Length of time of the sweep in seconds

void setup(void) {
  Serial.begin(9600);
  //while (!Serial);
```
Now, it's time to upload the code to your Trellis M4 and test it out! First, make sure you have headphones or speakers plugged into the Trellis M4.

Next, plug the Trellis M4 into your computer with the USB cable (make sure it's a data cable, not an evil power-only cable!)
Bootloader Mode

Now, we'll put the Trellis M4 into "bootloader" mode. In this mode it will be ready to receive the Arduino upload. Use a thin, pointed object such as a headphone plug or chopstick to double-click the reset button on the back side of the board.

Once you have double-clicked the reset button, the indicator LED will turn green. (You'll notice a new USB drive appear on your computer named TRELM4BOOT, this is the bootloader USB storage built right into the Trellis M4. We don't need to use this for our Arduino sketch upload.) It is now ready to receive the Arduino upload.
In Arduino, select the Trellis M4 board from the Tools > Board: menu item

Choose the proper port from the Tools > Port menu item as shown

Now, compile and upload the code to the Trellis M4 by clicking Sketch > Upload

You will hear a long upward and then downward tone sweep. Success!

Waveform Pitch

You can make a simple patch that swaps the tone sweep for a waveform node in the Audio System Design Tool in order to test the Trellis M4 hardware while playing a sound. The buttons will light up and the accelerometer tilt will bend the pitch (frequency) of the waveform oscillator.
First, copy and save this code as a new Arduino sketch.

```cpp
// Trellis M4 Audio Workshop
// shows how to alter pitch with accelerometer
// Waveform Mod

#include <Audio.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL343.h>
#include "Adafruit_NeoTrellisM4.h"
#include &lt;elapsedMillis.h&gt;

Adafruit_ADXL343 accel = Adafruit_ADXL343(123, &Wire1);

Adafruit_NeoTrellisM4 trellis = Adafruit_NeoTrellisM4();

// The NeoTrellisM4 object is a keypad and neopixel strip subclass
// that does things like auto-update the NeoPixels and stuff!
Adafruit_NeoTrellisM4 trellis = Adafruit_NeoTrellisM4();
// Paste your Audio System Design Tool code below this line:

int xbend = 64;
int ybend = 64;
int last_xbend = 64;
int last_ybend = 64;
int count=1;

void setup() {
  trellis.begin();
  trellis.show();  // Initialize w all pixels off
  trellis.setBrightness(255);

  if(!accel.begin()) {
    Serial.println("No accelerometer found");
    while(1);
  }

  AudioMemory(10);
  // Initialize processor and memory measurements
  AudioProcessorUsageMaxReset();
  AudioMemoryUsageMaxReset();
  Serial.begin(115200);
  waveform1.begin(WAVEFORM_SAWTOOTH);
  delay(1000);
}

void loop() {
  waveform1.frequency(110 + (ybend * 2));
  waveform1.amplitude(0.05);
  wait(5);
}

void wait(unsigned int milliseconds){
  elapsedMillis msec=0;
}
while (msec &lt;= milliseconds) {
    trellis.tick();
    while (trellis.available()) {
        keypadEvent e = trellis.read();
        Serial.print((int)e.bit.KEY);
        int keyindex = e.bit.KEY;
        if (e.bit.EVENT == KEY_JUST_PRESSED) {
            Serial.println(" pressed");
            trellis.setPixelColor(keyindex, Wheel(keyindex * 255 / 32)); // rainbow!
        } else if (e.bit.EVENT == KEY_JUST_RELEASED) {
            Serial.println(" released");
            trellis.setPixelColor(keyindex, 0);
        }
    }

    // Check for accelerometer
    sensors_event_t event;
    accel.getEvent(&amp;event);

    // check if it's been moved a decent amount
    if (abs(event.acceleration.x) &lt; 2.0) { // 2.0 m/s^2
        // don't make any bend unless they've really started moving it
        xbend = 64;
    } else {
        if (event.acceleration.x &gt; 0) {
            xbend = ofMap(event.acceleration.x, 2.0, 10.0, 63, 0, true); // 2 - 10 m/s^2
            is upward bend
        } else {
            xbend = ofMap(event.acceleration.x, -2.0, -10.0, 64, 127, true); // -2 ~ -10 m/s^2 is downward bend
        }
    }
    if (xbend != last_xbend) {
        Serial.print("X mod: "); Serial.println(xbend);
        last_xbend = xbend;
    }

    if (abs(event.acceleration.y) &lt; 2.0) { // 2.0 m/s^2
        ybend = 64;
    } else {
        if (event.acceleration.y &gt; 0) {
            ybend = ofMap(event.acceleration.y, 2.0, 10.0, 63, 0, true); // 2 ~ 10 m/s^2
            is upward bend
        } else {
            ybend = ofMap(event.acceleration.y, -2.0, -10.0, 64, 127, true); // -2 ~ -10 m/s^2 is downward bend
        }
    }
    if (ybend != last_ybend) {
        Serial.print("Y mod: "); Serial.println(ybend);
        last_ybend = ybend;
    }
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
    WheelPos = 255 - WheelPos;
    if(WheelPos &lt; 85) {
        return Adafruit_NeoPixel::Color(255 - WheelPos * 3, 0, WheelPos * 3);
    }
    if(WheelPos &lt; 170) {
WheelPos -= 85;
    return Adafruit_NeoPixel::Color(0, WheelPos * 3, 255 - WheelPos * 3);
}
WheelPos -= 170;
    return Adafruit_NeoPixel::Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}

// floating point map
float ofMap(float value, float inputMin, float inputMax, float outputMin, float outputMax, bool clamp) {
    float outVal = ((value - inputMin) / (inputMax - inputMin) * (outputMax - outputMin) + outputMin);
    if (clamp) {
        if (outputMax < outputMin) {
            if (outVal < outputMax) outVal = outputMax;
            else if (outVal > outputMin) outVal = outputMin;
        } else {
            if (outVal > outputMax) outVal = outputMax;
            else if (outVal < outputMin) outVal = outputMin;
        }
    }
    return outVal;
}

You can see that we're now using some of the built in NeoTrellis M4 library functions to read buttons, light the button NeoPixels, and read the accelerometer.

We've got some helper functions to make this easier, such as the Wheel function for picking colors, and the ofMap function to map the accelerometer values from floating point numbers to integers.

Waveform

The key elements of the Audio library code here are: waveform1.begin(WAVEFORM_SAWTOOTH); which initializes the waveform1 node as a sawtooth waveform in the setup().

Then, in order to adjust the pitch of the waveform interactively, we'll make a waveform1.frequency() call in the main loop() and give it a value of 110 + a value mapped to the accelerometer reading on the y-axis, which looks like this: waveform1.frequency(110 + (ybend * 2));

The Info panel in the Audio System Design Tool shows us the different functions available:

Functions

begin(waveform);
Configure the waveform type to create.

`begin(level, frequency, waveform);`

Output a waveform, and set the amplitude and frequency.

`frequency(freq);`

Change the frequency.

`amplitude(level);`

Change the amplitude. Set to 0 to turn the signal off.

`offset(level);`

Add a DC offset, from -1.0 to +1.0. Useful for generating waveforms to use as control or modulation signals.

`phase(angle);`

Cause the generated waveform to jump to a specific point within its cycle. Angle is from 0 to 360 degrees. When multiple objects are configured, `AudioNoInterrupts()` should be used to guarantee all new settings take effect together.

`pulseWidth(amount);`

Change the width (duty cycle) of the pulse.

`arbitraryWaveform(array, maxFreq);`

Configure the waveform to be used with WAVEFORM_ARBITRARY. Array must be an array of 256 samples. Currently, the data is used without any filtering, which can cause aliasing with frequencies above 172 Hz. For higher frequency output, you must bandwidth limit your waveform data. Someday, "maxFreq" will be used to do this automatically.
Notes

Supported Waveforms:

- WAVEFORM_SINE
- WAVEFORM_SAWTOOTH
- WAVEFORM_SAWTOOTH_REVERSE
- WAVEFORM_SQUARE
- WAVEFORM_TRIANGLE
- WAVEFORM_TRIANGLE_VARIABLE
- WAVEFORM_ARBITRARY
- WAVEFORM_PULSE
- WAVEFORM_SAMPLE_HOLD

Waveform to Audio Output

In the Audio System Design Tool, create a waveform node and connect it to a dacs node renamed to "audioOutput", then copy the GUI code snippet.
Paste the code snippet into your Arduino sketch and save it as `Waveform_Mode_TrellisM4.ino`

Upload the code to your board. Now, you'll be able to bend the pitch by tilting the NeoTrellis, and the buttons will light up when pressed.

This is what the final sketch looks like after adding the Audio GUI code:

```cpp
// SPDX-FileCopyrightText: 2018 John Park for Adafruit Industries
// SPDX-License-Identifier: MIT
// Trellis M4 Audio Workshop
// shows how to alter pitch with accelerometer
// Waveform Mod
#include <Audio.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL343.h>
#include "Adafruit_NeoTrellisM4.h"
#include <elapsedMillis.h>

Adafruit_ADXL343 accel = Adafruit_ADXL343(123, &Wire1);

// The NeoTrellisM4 object is a keypad and neopixel strip subclass
// that does things like auto-update the NeoPixels and stuff!
Adafruit_NeoTrellisM4 trellis = Adafruit_NeoTrellisM4();
// Paste your Audio System Design Tool code below this line:
// GUItool: begin automatically generated code
AudioSynthWaveform waveform1; // xy=592.7221984863281,187.38888549804688
AudioOutputAnalogStereo audioOutput; // xy=777.0833129882812,189.08334350585938
AudioConnection patchCord1(waveform1, 0, audioOutput, 0);
AudioConnection patchCord2(waveform1, 0, audioOutput, 1);
// GUItool: end automatically generated code

int xbend = 64;
int ybend = 64;
int last_xbend = 64;
int last_ybend = 64;

int count = 1;

void setup() {
  trellis.begin();
  trellis.show(); // Initialize w all pixels off
  trellis.setBrightness(255);

  if(!accel.begin()) {
    Serial.println("No accelerometer found");
    while(1);
  }
  AudioMemory(10);
  // Initialize processor and memory measurements
  AudioProcessorUsageMaxReset();
  AudioMemoryUsageMaxReset();
  Serial.begin(115200);
  waveform1.begin(WAVEFORM_SAWTOOTH);
  delay(10000);
}
```
void loop() {
  waveform1.frequency(110 + (ybend * 2));
  waveform1.amplitude(0.05);
  wait(5);
}

void wait(unsigned int milliseconds){
  elapsedMillis msec = 0;
  while (msec <= milliseconds){
     trellis.tick();
     while (trellis.available()) {
        keypadEvent e = trellis.read();
        Serial.print((int)e.bit.KEY);
        int keyindex = e.bit.KEY;
        if (e.bit.EVENT == KEY_JUST_PRESSED){
          Serial.println(" pressed");
          trellis.setPixelColor(keyindex, Wheel(keyindex * 255 / 32)); // rainbow!
        } else if (e.bit.EVENT == KEY_JUST_RELEASED){
          Serial.println(" released");
          trellis.setPixelColor(keyindex, 0);
        }
    }
  }

  // Check for accelerometer
  sensors_event_t event;
  accel.getEvent(&event);

  //check if it's been moved a decent amount
  if (abs(event.acceleration.x) < 2.0) { // 2.0 m/s^2
    // don't make any bend unless they've really started moving it
    xbend = 64;
  } else {
    if (event.acceleration.x > 0) {
      xbend = ofMap(event.acceleration.x, 2.0, 10.0, 63, 0, true); // 2 ~ 10 m/s^2
      is upward bend
    } else {
      xbend = ofMap(event.acceleration.x, -2.0, -10.0, 64, 127, true); // -2 ~ -10 m/s^2 is downward bend
    }
  }
  if (xbend != last_xbend) {
    Serial.print("X mod: "); Serial.println(xbend);
    last_xbend = xbend;
  }

  if (abs(event.acceleration.y) < 2.0) { // 2.0 m/s^2
    ybend = 64;
  } else {
    if (event.acceleration.y > 0) {
      ybend = ofMap(event.acceleration.y, 2.0, 10.0, 63, 0, true); // 2 ~ 10 m/s^2
      is upward bend
    } else {
      ybend = ofMap(event.acceleration.y, -2.0, -10.0, 64, 127, true); // -2 ~ -10 m/s^2 is downward bend
    }
  }
  if (ybend != last_ybend) {
    Serial.print("Y mod: "); Serial.println(ybend);
    last_ybend = ybend;
  }
}
// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
    WheelPos = 255 - WheelPos;
    if (WheelPos < 85) {
        return Adafruit_NeoPixel::Color(255 - WheelPos * 3, 0, WheelPos * 3);
    }
    if (WheelPos < 170) {
        WheelPos -= 85;
        return Adafruit_NeoPixel::Color(0, WheelPos * 3, 255 - WheelPos * 3);
    }
    WheelPos -= 170;
    return Adafruit_NeoPixel::Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}

// floating point map
float ofMap(float value, float inputMin, float inputMax, float outputMin, float outputMax, bool clamp) {
    float outVal = ((value - inputMin) / (inputMax - inputMin)) * (outputMax - outputMin) + outputMin;
    if (clamp) {
        if (outputMax < outputMin) {
            if (outVal < outputMax) outVal = outputMax;
        } else if (outVal > outputMin) outVal = outputMin;
        else {
            if (outVal > outputMax) outVal = outputMax;
            else if (outVal < outputMin) outVal = outputMin;
        }
    }
    return outVal;
}

You can also open the Arduino Serial Monitor to see the print statements of the accelerometer readings and button presses. This is very useful for troubleshooting!

First, re-select the TrellisM4 in the Tools > Port menu (a new port is often assigned after upload), then open the serial monitor by clicking Tools > Serial Monitor

Next, we'll create a more sophisticated, four voice polyphonic synthesizer.
Full Synth Demo

Four Oscillator Synthesizer

This final example will give you an idea of how to patch a more complex synthesizer that uses four waveform nodes, each initialized to a different waveform type.

These will each flow through a different envelope, and in some cases their own effects nodes.

Finally, we'll use multiple mixers to combine the sounds together.

To begin, replicate the patch shown above.

Most of it is pretty straightforward, with four waveforms each running through a respective envelope.

mixer1 is used to combine the four waveforms, with some of them running through effects first.

We then use a pair of mixers, mixerLeft and mixerRight to mix in the multiple taps of the delay, and then combine everything for stereo audioOut.

These are the relevant settings used to make this sounds come alive. (You can experiment with them once you've got it running later.)
Waveforms

We're setting up the four waveform nodes as sine, square, sawtooth, and triangle waves respectively.

```cpp
wave0.begin(0.85, 50, WAVEFORM_SINE);
wave1.begin(0.4, 50, WAVEFORM_SQUARE);
wave2.begin(0.6, 50, WAVEFORM_SAWTOOTH);
wave3.begin(0.4, 50, WAVEFORM_TRIANGLE);
```

Envelopes

The envelopes are defined with the attack, hold, decay, sustain, and release settings. Our first row has a very soft attack and a long release:

```cpp
env0.attack(300);
env0.hold(2);
env0.decay(30);
env0.sustain(0.6);
env0.release(1200);
```

The second oscillator has a quick attack and a moderate release:

```cpp
env1.attack(10);
env1.hold(2);
env1.decay(30);
env1.sustain(0.6);
env1.release(400);
```

The third oscillator is similar to the second with a longer release:

```cpp
env2.attack(10);
env2.hold(20);
env2.decay(30);
env2.sustain(0.6);
env2.release(1000);
```

The fourth oscillator has a quick attack and moderately long release:

```cpp
env3.attack(10);
env3.hold(2);
```
Arduino Code

This is the code you’ll copy and paste into a new Arduino sketch.

```c
/* Audio library demonstration - pocket synth with C major scale and 4 wave types */
//each row is a different waveform, envelope, and effect set in major scale
// row 0 sine, soft attack, long release ADSR
// row 1 square, hard attack, moderate release, flanger effect
// row 2 sawtooth, hard attack, soft release, chorus effect
// row 3 triangle, medium attack, long release ADSR, multi tap delay

#include &lt;Audio.h&gt;
#include &lt;Adafruit_NeoTrellisM4.h&gt;

Adafruit_NeoTrellisM4 trellis = Adafruit_NeoTrellisM4();
// Paste your Audio System Design Tool code below this line:

AudioSynthWaveform *waves[4] = {
  &amp;wave0, &amp;wave1, &amp;wave2, &amp;wave3,
};
short wave_type[4] = {
  WAVEFORM_SINE,
  WAVEFORM_SQUARE,
  WAVEFORM_SAWTOOTH,
  WAVEFORM_TRIANGLE,
};
float cmaj_low[8] = { 130.81, 146.83, 164.81, 174.61, 196.00, 220.00, 246.94, 261.63 };
float cmaj_high[8] = { 261.6, 293.7, 329.6, 349.2, 392.0, 440.0, 493.9, 523.3 };
AudioEffectEnvelope *envs[4] = {
  &amp;env0, &amp;env1, &amp;env2, &amp;env3,
};
int n_chorus = 5;
#define CHORUS_DELAY_LENGTH (400*AUDIO_BLOCK_SAMPLES)
short chorusDelayline[CHORUS_DELAY_LENGTH];
#define FLANGER_DELAY_LENGTH (6*AUDIO_BLOCK_SAMPLES)
short flangerDelayline[FLANGER_DELAY_LENGTH];

void setup(){
  Serial.begin(115200);
  //while (!Serial);

  trellis.begin();
  trellis.setBrightness(255);

  AudioMemory(120);

  //Initialize the waveform nodes
  wave0.begin(0.85, 50, WAVEFORM_SINE);
  wave1.begin(0.4, 50, WAVEFORM_SQUARE);
```
wave2.begin(0.6, 50, WAVEFORM_SAWTOOTH);
wave3.begin(0.4, 50, WAVEFORM_TRIANGLE);

// reduce the gain on some channels, so half of the channels
// are "positioned" to the left, half to the right, but all
// are heard at least partially on both ears
mixerLeft.gain(0, 0.3);
mixerLeft.gain(1, 0.1);
mixerLeft.gain(2, 0.5);
mixerRight.gain(0, 0.3);
mixerRight.gain(1, 0.5);
mixerRight.gain(2, 0.1);

// set envelope parameters, for pleasing sound :-)
env0.attack(300);
env0.hold(2);
env0.decay(30);
env0.sustain(0.6);
env0.release(1200);
env1.attack(10);
env1.hold(2);
env1.decay(30);
env1.sustain(0.6);
env1.release(400);
env2.attack(10);
env2.hold(20);
env2.decay(30);
env2.sustain(0.6);
env2.release(1000);
env3.attack(10);
env3.hold(2);
env3.decay(30);
env3.sustain(0.6);
env3.release(600);

// set delay parameters
delay1.delay(0, 110);
delay1.delay(1, 660);
delay1.delay(2, 220);
delay1.delay(3, 1220);

// set effects parameters
chorus1.begin(chorusDelayline, CHORUS_DELAY_LENGTH, n_chorus);
flange1.begin(flangerDelayline, FLANGER_DELAY_LENGTH, FLANGER_DELAY_LENGTH/4,
FLANGER_DELAY_LENGTH/4, .5);

Serial.println("setup done");

// Initialize processor and memory measurements
AudioProcessorUsageMaxReset();
AudioMemoryUsageMaxReset();
}

void noteOn(int num){

int voice = num/8;
float *scale;
if(voice == 0 || voice == 1) scale = cmaj_low;
else scale = cmaj_high;
AudioNoInterrupts();
//waves[voice]-&gt;begin(.5, scale[num%8], wave_type[voice]);
waves[voice]-&gt;frequency(scale[num%8]);
envs[voice]-&gt;noteOn();
AudioInterrupts();
}
void noteOff(int num){
    int voice = num/8;
    envs[voice]->noteOff();
}

void loop() {
    trellis.tick();

    while(trellis.available())
    {
        keypadEvent e = trellis.read();
        int keyindex = e.bit.KEY;
        if(e.bit.EVENT == KEY_JUST_PRESSED){
            //trellis.setPixelColor(keyindex, 0xFFFFFFFF); // plain white
            trellis.setPixelColor(keyindex, Wheel(keyindex * 255 / 32)); // rainbow!
            noteOn(keyindex);
        }
        else if(e.bit.EVENT == KEY_JUST_RELEASED){
            noteOff(keyindex);
            trellis.setPixelColor(keyindex, 0);
        }
    }
    delay(10);
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
    WheelPos = 255 - WheelPos;
    if(WheelPos < 85) {
        return Adafruit_NeoPixel::Color(255 - WheelPos * 3, 0, WheelPos * 3);
    }
    if(WheelPos < 170) {
        WheelPos -= 85;
        return Adafruit_NeoPixel::Color(0, WheelPos * 3, 255 - WheelPos * 3);
    }
    WheelPos -= 170;
    return Adafruit_NeoPixel::Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}

Now, export the source code snippet from the Audio System Design Tool and paste it into the section indicated in the Arduino sketch.

Save this code as Synth_Design_Tool_Demo_TrellisM4.ino and then upload it to the board.

This is what the final code looks like with the Audio System Design Tool GUI code added.

// SPDX-FileCopyrightText: 2018 John Park for Adafruit Industries
//
// SPDX-License-Identifier: MIT
/* Audio library demonstration - pocket synth with C major scale and 4 wave types */
//each row is a different waveform, envelope, and effect set in major scale
// row 0 sine, soft attack, long release ADSR
// row 1 square, hard attack, moderate release, flanger effect
// row 2 sawtooth, hard attack, soft release, chorus effect
// row 3 triangle, medium attack, long release ADSR, multi tap delay
#include <Audio.h>
#include <Adafruit_NeoTrellisM4.h>

Adafruit_NeoTrellisM4 trellis = Adafruit_NeoTrellisM4();

// Paste your Audio System Design Tool code below this line:

AudioSynthWaveform wave0;
AudioSynthWaveform wave1;
AudioSynthWaveform wave2;
AudioSynthWaveform wave3;
AudioEffectEnvelope env0;
AudioEffectEnvelope env1;
AudioEffectEnvelope env2;
AudioEffectEnvelope env3;
AudioEffectChorus chorus1;
AudioEffectFlange flange1;
AudioEffectDelay delay1;
AudioMixer4 mixer1;
AudioMixer4 mixerLeft;
AudioMixer4 mixerRight;
AudioOutputAnalogStereo audioOut;

AudioConnection patchCord1(wave0, env0);
AudioConnection patchCord2(wave1, env1);
AudioConnection patchCord3(wave2, env2);
AudioConnection patchCord4(wave3, env3);
AudioConnection patchCord5(env0, mixer1, 0);
AudioConnection patchCord6(env1, flange1);
AudioConnection patchCord7(env2, chorus1);
AudioConnection patchCord8(env3, delay1);
AudioConnection patchCord9(env3, mixer1, 3);
AudioConnection patchCord10(chorus1, 0, mixer1, 2);
AudioConnection patchCord11(flange1, 0, mixer1, 1);
AudioConnection patchCord12(delay1, 0, mixerLeft, 1);
AudioConnection patchCord13(delay1, 1, mixerLeft, 2);
AudioConnection patchCord14(delay1, 2, mixerRight, 1);
AudioConnection patchCord15(delay1, 3, mixerRight, 2);
AudioConnection patchCord16(mixer1, 0, mixerLeft, 0);
AudioConnection patchCord17(mixer1, mixerLeft, 0);
AudioConnection patchCord18(mixerLeft, 0, audioOut, 0);
AudioConnection patchCord19(mixerRight, 0, audioOut, 1);

AudioSynthWaveform *waves[4] = {
  &wave0, &wave1, &wave2, &wave3,
};
short wave_type[4] = {
  WAVEFORM_SINE, WAVEFORM_SQUARE, WAVEFORM_SAWTOOTH, WAVEFORM_TRIANGLE,
};
float cmaj_low[8] = { 130.81, 146.83, 164.81, 174.61, 196.00, 220.00, 246.94, 261.63 };
float cmaj_high[8] = { 261.6, 293.7, 329.6, 349.2, 392.0, 440.0, 493.9, 523.3 };
AudioEffectEnvelope *envs[4] = {
  &env0, &env1, &env2, &env3,
};

int n_chorus = 5;
#define CHORUS_DELAY_LENGTH (400*AUDIO_BLOCK_SAMPLES)
short chorusDelayLine[CHORUS_DELAY_LENGTH];
#define FLANGER_DELAY_LENGTH (6*AUDIO_BLOCK_SAMPLES)
short flangerDelayLine[FLANGER_DELAY_LENGTH];

void setup(){
  Serial.begin(115200);
  //while (!Serial);

  ©Adafruit Industries
  Page 54 of 56
trellis.begin();
trellis.setBrightness(255);

AudioMemory(120);

// Initialize the waveform nodes
wave0.begin(0.85, 50, WAVEFORM_SINE);
wave1.begin(0.4, 50, WAVEFORM_SQUARE);
wave2.begin(0.6, 50, WAVEFORM_SAWTOOTH);
wave3.begin(0.4, 50, WAVEFORM_TRIANGLE);

// reduce the gain on some channels, so half of the channels
// are "positioned" to the left, half to the right, but all
// are heard at least partially on both ears
mixerLeft.gain(0, 0.3);
mixerLeft.gain(1, 0.1);
mixerLeft.gain(2, 0.5);
mixerRight.gain(0, 0.3);
mixerRight.gain(1, 0.5);
mixerRight.gain(2, 0.1);

// set envelope parameters, for pleasing sound :-)  
env0.attack(300);
env0.hold(2);
env0.decay(30);
env0.sustain(0.6);
env0.release(1200);

env1.attack(10);
env1.hold(2);
env1.decay(30);
env1.sustain(0.6);
env1.release(400);

env2.attack(10);
env2.hold(20);
env2.decay(30);
env2.sustain(0.6);
env2.release(1000);

env3.attack(10);
env3.hold(2);
env3.decay(30);
env3.sustain(0.6);
env3.release(600);

// set delay parameters
delay1.delay(0, 110);
delay1.delay(1, 660);
delay1.delay(2, 220);
delay1.delay(3, 1220);

// set effects parameters
chorus1.begin(chorusDelayline, CHORUS_DELAY_LENGTH, n_chorus);
flange1.begin(flangerDelayline, FLANGER_DELAY_LENGTH, FLANGER_DELAY_LENGTH/4, FLANGER_DELAY_LENGTH/4, .5);

Serial.println("setup done");

// Initialize processor and memory measurements
AudioProcessorUsageMaxReset();
AudioMemoryUsageMaxReset();
}

void noteOn(int num){
  int voice = num/8;
float *scale;
if (voice == 0 || voice == 1) scale = cmaj_low;
else scale = cmaj_high;
AudioNoInterrupts();
waves[voice]->frequency(scale[num%8]);
envs[voice]->noteOn();
AudioInterrupts();
}

void noteOff(int num){
  int voice = num/8;
envs[voice]->noteOff();
}

void loop() {
  trellis.tick();
  while (trellis.available()) {
    keypadEvent e = trellis.read();
    int keyindex = e.bit.KEY;
    if (e.bit.EVENT == KEY_JUST_PRESSED){
      trellis.setPixelColor(keyindex, Wheel(keyindex * 255 / 32)); // rainbow!
      noteOn(keyindex);
    } else if (e.bit.EVENT == KEY_JUST_RELEASED){
      noteOff(keyindex);
      trellis.setPixelColor(keyindex, 0);
    }
    delay(10);
  }

  // Input a value 0 to 255 to get a color value.
  // The colours are a transition r - g - b - back to r.
  uint32_t Wheel(byte WheelPos) {
    WheelPos = 255 - WheelPos;
    if (WheelPos < 85) {
      return Adafruit_NeoPixel::Color(255 - WheelPos * 3, 0, WheelPos * 3);
    }
    if (WheelPos < 170) {
      WheelPos -= 85;
      return Adafruit_NeoPixel::Color(0, WheelPos * 3, 255 - WheelPos * 3);
    }
    WheelPos -= 170;
    return Adafruit_NeoPixel::Color(WheelPos * 3, 255 - WheelPos * 3, 0);
  }

Now, you can play the synth! Each row plays a C major scale using a different oscillator waveform type or mix. Try playing notes on multiple rows to create chords.