Mini LED Matrix Audio Visualizer

Created by Liz Clark

https://learn.adafruit.com/mini-led-matrix-audio-visualizer

Last updated on 2022-03-02 09:56:02 AM EST
Table of Contents

Overview 3
- Prerequisite Guides 4
- Parts 4

Circuit Diagram 6
- Adafruit Library for Fritzing 6
- Wiring the IS31FL3741 Display to the Adafruit Feather Sense 6

CircuitPython on Feather Sense 6
- Set up CircuitPython Quick Start! 7

Code the Waterfall Visualizer 9
- Upload the Code and Libraries to the Feather nRF52840 Sense 11
- How the CircuitPython Code Works 12

Code the Audio Spectrum Light Show 14
- Upload the Code and Libraries to the Feather nRF52840 Sense 17

3D Printing 18
- CAD Parts List 18
- CAD Assembly 18
- Slicing Parts 19
- Design Source Files 19

Assembly 19
- Hardware for PCBs 19
- Install LED Matrix 20
- Secure LED Matrix 20
- Install and Secure Feather 21
- Install PCB Mount to Case 21
- Installed PCB Mount 22
- Secure PCB Mount to Case 22
- Install Cover 22
- Final Build 23
Overview

In this project, you'll learn how to build an audio reactive LED matrix using the 13x9 IS31FL3741 LED matrix and Feather Sense nRF52840 board with CircuitPython.

The Feather Sense nRF52840 features a built-in PDM MEMs microphone for audio input. Audio is sampled via the onboard PDM microphone. The sampled data is then visualized with color.

The 3D printed case houses the two boards and snap fits for an easy assembly. The case features a built-in button presser for using the reset button without having to take it apart. The LED matrix and Feather are secured to a PCB bracket with hardware screws and nuts.
Prerequisite Guides

Take a moment to review the following guides.

- Adafruit Feather nRF52840 Sense (https://adafru.it/L6d)
- Adafruit IS31FL3741 (https://adafru.it/Vcr)
- ulab: Crunch numbers fast in CircuitPython (https://adafru.it/KaJ)

Parts

Adafruit Feather nRF52840 Sense
The Adafruit Feather Bluefruit Sense takes our popular Feather nRF52840 Express and adds a smorgasbord of sensors...
https://www.adafruit.com/product/4516
Adafruit IS31FL3741 13x9 PWM RGB LED Matrix Driver
Add a splash of RGB LEDs to a project you're working on, with this adorable 13x9 RGB LED matrix breakout. It features -- no surprise -- 117 RGB LEDs, each one 2x2mm in size, in a...
https://www.adafruit.com/product/5201

STEMMA QT / Qwiic JST SH 4-pin to Premium Male Headers Cable
This 4-wire cable is a little over 150mm / 6" long and fitted with JST-SH female 4-pin connectors on one end and premium Dupont male headers on the other. Compared with the...
https://www.adafruit.com/product/4209

Silicone Cover Stranded-Core Ribbon Cable - 10 Wire 1 Meter Long
For those who are fans of our silicone-covered wires, but are always looking to up their wiring game. We now have Silicone Cover Ribbon cables! These may look...
https://www.adafruit.com/product/3890

Black Nylon Machine Screw and Stand-off Set – M2.5 Thread
Totaling 380 pieces, this M2.5 Screw Set is a must-have for your workstation. You'll have enough screws, nuts, and hex standoffs to fuel your maker...
https://www.adafruit.com/product/3299
Circuit Diagram

The diagram below provides a visual reference for wiring of the components. This diagram was created using the software package [Fritzing](https://adafru.it/oEP).

Adafruit Library for Fritzing

Use Adafruit's Fritzing parts library to create circuit diagrams for your projects. Download the library or just grab individual parts. Get the library and parts from [GitHub - Adafruit Fritzing Parts](https://adafru.it/AYZ).

[Image: Circuit Diagram]

Wiring the IS31FL3741 Display to the Adafruit Feather Sense

- SDA from Feather to SDA on IS31FL3741
- SCL from Feather to SCL on IS31FL3741
- 3V from Feather to VCC on IS31FL3741
- GND from Feather to GND on IS31FL3741

CircuitPython on Feather Sense

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

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)

Download the latest version of CircuitPython for this board via circuitpython.org
https://adafru.it/JqE

Click the link above to download the latest UF2 file.

Download and save it to your desktop (or wherever is handy).
Plug your Feather Sense into your computer using a known-good USB cable.

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

Double-click the Reset button next to the USB connector on your board, and you will see the NeoPixel RGB LED turn green (identified by the arrow in the image). If it turns red, check the USB cable, try another USB port, etc. Note: The little red LED next to the USB connector will pulse red. That's ok!

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

You will see a new disk drive appear called FTHRSNSBOOT.

Drag the adafruit_circuitpython_etc.uf2 file to FTHRSNSBOOT.
The LED will flash. Then, the FTHRSNSBOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it, you're done! :)

Note: Some early release Sense boards had the drive named FTHR840BOOT. You can still copy .UF2s to the board, just copy to the board name appearing when the board is plugged in.

Code the Waterfall Visualizer

Once you've finished setting up your Feather nRF52840 Sense with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

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

```python
# SPDX-FileCopyrightText: 2021 Liz Clark for Adafruit Industries
# SPDX-License-Identifier: MIT
```
from adafruit_is31fl3741.adafruit_rgbmatrixqt import Adafruit_RGBMatrixQT

#  Manually declare I2c (not board.I2C()) to access 1 MHz speed for
i2c = busio.I2C(board.SCL, board.SDA, frequency=1000000)
#  Declare is31 w/buffering preferred (low RAM will fall back on unbuffered)
is31 = Adafruit_RGBMatrixQT(i2c, allocate=adafruit_is31fl3741.PREFER_BUFFER)
#  In buffered mode, MUST use show() to refresh matrix (see line 94)

#  brightness for the RGBMatrixQT
#  set to about 20%
is31.set_led_scaling(0x19)
is31.global_current = 0x03
is31.enable = True

heatmap = [0xb000ff, 0xa600ff, 0x9b00ff, 0x8f00ff, 0x8200ff, 0x7400ff, 0x6500ff, 0x5200ff, 0x3900ff, 0x0003ff, 0x0003ff, 0x0047ff, 0x0066ff, 0x007eff, 0x0093ff, 0x00a6ff, 0x00b7ff, 0x00c8ff, 0x00d7ff, 0x00e5ff, 0x00e0ff, 0x00e6fd, 0x00ecf6, 0x00f2ea, 0x00f6d7, 0x00fac0, 0x00fca3, 0x00fe81, 0x00ff59, 0x00ff16, 0x08ff00, 0x82ff00, 0x78ff00, 0x8bff00, 0x9bff00, 0xaaff00, 0xb8ff00, 0xc5ff00, 0xd1ff00, 0xedff00, 0xf5eb00, 0x00c100, 0xffab00, 0xff9500, 0xff7c00, 0xff6100, 0xff4100, 0xff0000, 0xff0000, 0xff0000]
mic.record(samples_bit, len(samples_bit))
# send the sample to the ulab array
samples = np.array(samples_bit[3:3])
# creates a spectogram of the data
spectrogram1 = np.log(spectrogram(samples))
# spectrum() is always nonnegative, but add a tiny value
# to change any zeros to nonzero numbers
spectrogram1 = np.log(spectrogram1 + 1e-7)
spectrogram1 = spectrogram1[:,(fft_size//2)-1]
# sets range of the spectrogram
min_curr = np.min(spectrogram1)
max_curr = np.max(spectrogram1)
# resets values
if max_curr > max_all:
  max_all = max_curr
else:
  max_curr = max_curr - 1
min_curr = max(min_curr, 3)
# stores spectrogram in data
data = (spectrogram1 - min_curr) * (51. / (max_all - min_curr))
# sets negative numbers to zero
data = data * np.array((data > 0))
# resets y
y = scroll_offset
# runs waves to write data to the LED's
waves(data, y)
# updates scroll offset to move data along matrix
scroll_offset = (y + 1) % 9
# writes data to the RGB matrix
is31.show()

main()

Upload the Code and Libraries to the Feather nRF52840 Sense

After downloading the Project Bundle, plug your Feather nRF52840 Sense into the computer's USB port. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called CIRCUITPY. Unzip the folder and copy the following items to the Feather nRF52840 Sense's CIRCUITPY drive.

- lib folder
- code.py

Your Feather nRF52840 Sense CIRCUITPY drive should look like this after copying the lib folder and the code.py file.
How the CircuitPython Code Works

The code is a port of the Waterfall Spectrum Analyzer CircuitPython code (https://adafru.it/Vcs) written by Jeff Epler for the ulab: Crunch Numbers Fast in CircuitPython Learn Guide (https://adafru.it/Vct). Be sure to check out that guide for more information on the ulab CircuitPython library.

The main portion of the code remains the same. Audio is sampled via the onboard PDM microphone. The sampled data is then visualized with color. The difference is that instead of displaying that visualization with a screen, as seen on the CLUE board, it is displayed with the LED's on the matrix.

```python
# setup for onboard mic
mic = audiobusio.PDMIn(board.MICROPHONE_CLOCK, board.MICROPHONE_DATA,
sample_rate=16000, bit_depth=16)

# use some extra sample to account for the mic startup
samples_bit = array.array('H', [0] * (fft_size+3))

# sends visualized data to the RGB matrix with colors
def waves(data, y):
    offset = max(0, (13-len(data))//2)
    for x in range(min(13, len(data))):
        is31.pixel(x+offset, y, heatmap[int(data[x])])
```

In the loop, you’ll find the ulab CircuitPython library in action, using numpy to crunch the sampled data into a pretty, colorful light show.

```python
while True:
    # record the audio sample
```
mic.record(samples_bit, len(samples_bit))
# send the sample to the ulab array
samples = np.array(samples_bit[3:3])
# creates a spectogram of the data
spectrogram1 = spectrogram(samples)
# spectrum() is always nonnegative, but add a tiny value
# to change any zeros to nonzero numbers
spectrogram1 = np.log(spectrogram1 + 1e-7)
spectrogram1 = spectrogram1[fft_size//2:-1]
# sets range of the spectogram
min_curr = np.min(spectrogram1)
max_curr = np.max(spectrogram1)
# resets values
if max_curr &gt; max_all:
    max_all = max_curr
else:
    max_curr = max_curr-1
min_curr = max(min_curr, 3)
# stores spectogram in data
data = (spectrogram1 - min_curr) * (51. / (max_all - min_curr))
# sets negative numbers to zero
data = data * np.array((data &gt; 0))
# resets y
y = scroll_offset
# runs waves to write data to the LED's
waves(data, y)
# updates scroll_offset to move data along matrix
scroll_offset = (y + 1) % 9
# writes data to the RGB matrix
is31.show()

Need for Speed

It's important for data visualizations to be as speedy and reactive as possible for the best effect. To do this for the IS31FL3741 matrix, I2C is overclocked to communicate using 1 MHz speed. Additionally, the FFT data sample size is set to 64.

# Manually declare I2c (not board.I2C()) to access 1 MHz speed for
i2c = busio.I2C(board.SCL, board.SDA, frequency=1000000)
# Declare is31 w/buffering preferred (low RAM will fall back on unbuffered)
is31 = Adafruit_RGBMatrixQT(i2c, allocate=adafruit_is31fl3741.PREFER_BUFFER)
# In buffered mode, MUST use show() to refresh matrix (see line 94)

# size of the FFT data sample
fft_size = 64

All the Colors

After speed, the colors used to visualize the data come into play. The selected gradient in the code is a rainbow effect ranging from purple to red through the color spectrum. You can experiment with creating your own gradients too. There are quite a few tools available online for this. For this project, [colordesigner.io](https://adafru.it/Vcu) was used.
The color gradient is stored in the `heatmap` array and consists of 52 hexadecimal color codes.

```python
# array of colors for the LEDs
# goes from purple to red
# gradient generated using https://colordesigner.io/gradient-generator
heatmap = [0xb000ff, 0xa600ff, 0x9b00ff, 0x8f00ff, 0x8200ff, 0x7400ff, 0x6500ff, 0x5200ff, 0x3900ff, 0x0003ff, 0x0003ff, 0x0047ff, 0x0066ff, 0x007eff, 0x0093ff, 0x00a6ff, 0x00b7ff, 0x00c8ff, 0x00d7ff, 0x00e5ff, 0x00e0ff, 0x00e6fd, 0x00ecf6, 0x00f2ea, 0x00f6d7, 0x00fac0, 0x00fca3, 0x00fe81, 0x00ff59, 0x00ff16, 0x08ff16, 0x45ff08, 0x62ff00, 0x78ff00, 0x8bff00, 0x9bff00, 0xaaff00, 0xb8ff00, 0xc5ff00, 0xd1ff00, 0xe5ff00, 0xe0ff00, 0xe6fd00, 0xecf600, 0xf2ea00, 0xf6d700, 0xfca300, 0xface00, 0xface00, 0xfed000, 0xff0000, 0xff0000, 0xff0000]
```

---

## Code the Audio Spectrum Light Show

Once you've finished setting up your Feather nRF52840 Sense with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

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

```python
# SPDX-FileCopyrightText: 2021 Phil Burgess for Adafruit Industries
# # SPDX-License-Identifier: MIT

"""
AUDIO SPECTRUM LIGHT SHOW for Adafruit Eyelights (LED Glasses + Driver).
Uses onboard microphone and a lot of math to react to music.
"""

from array import array
from math import log
from time import monotonic
from supervisor import import_module
import board
from audiobusio import PDMIn
from busio import I2C
import adafruit_is31fl3741
import adafruit_is31fl3741.adafruit_rgbmatrixqt
from rainbowio import colorwheel
from ulab import numpy as np
from ulab.scipy.signal import spectrogram

# FFT/SPECTRUM CONFIG ----

fft_size = 256  # Sample size for Fourier transform, MUST be power of two
spectrum_size = fft_size // 2  # Output spectrum is 1/2 of FFT result
# Bottom of spectrum tends to be noisy, while top often exceeds musical
# range and is just harmonics, so clip both ends off:
low_bin = 10  # Lowest bin of spectrum that contributes to graph
high_bin = 75  # Highest bin "
```
# HARDWARE SETUP

# Manually declare I2C (not board.I2C() directly) to access 1 MHz speed...
i2c = I2C(board.SCL, board.SDA, frequency=1000000)

# Initialize the IS31 LED driver, buffered for smoother animation
glasses = LED_Glasses(i2c, allocate=adafruit_is31fl3741.MUST_BUFFER)
glasses = Adafruit_RGBMatrixQT(i2c, allocate=adafruit_is31fl3741.MUST_BUFFER)
glasses.show()  # Clear any residue on startup
glasses.set_led_scaling(0xFF)
glasses.global_current = 5  # Not too bright please
glasses.enable = True

# Initialize mic and allocate recording buffer (default rate is 16 MHz)
mic = PDMIn(board.MICROPHONE_CLOCK, board.MICROPHONE_DATA, bit_depth=16)
rec_buf = array("H", [0] * fft_size)  # 16-bit audio samples

# FFT/SPECTRUM SETUP

# To keep the display lively, tables are precomputed where each column of
# the matrix (of which there are few) is the sum value and weighting of
# several bins from the FFT spectrum output (of which there are many).
# The tables also help visually linearize the output so octaves are evenly
# spaced, as on a piano keyboard, whereas the source spectrum data is
# spaced by frequency in Hz.
column_table = []
spectrum_bits = log(spectrum_size, 2)  # e.g. 7 for 128-bin spectrum
low_frac = log(low_bin, 2) / spectrum_bits
frac_range = log(high_bin, 2) / spectrum_bits - low_frac
for column in range(glasses.width):
    # Determine the lower and upper frequency range for this column, as
    # fractions within the scaled 0.0 to 1.0 spectrum range. 0.95 below
    # creates slight frequency overlap between columns, looks nicer.
    lower = low_frac + frac_range * (column / glasses.width * 0.95)
    upper = low_frac + frac_range * ((column + 1) / glasses.width)
    mid = (lower + upper) * 0.5  # Center of lower-to-upper range
    half_width = (upper - lower) * 0.5  # 1/2 of lower-to-upper range
    # Map fractions back to spectrum bin indices that contribute to column
    first_bin = int(2 ** (spectrum_bits * lower) + 1e-4)
    last_bin = int(2 ** (spectrum_bits * upper) + 1e-4)
    bin_weights = []  # Each spectrum bin's weighting will be added here
    for bin_index in range(first_bin, last_bin + 1):
        # Find distance from column's overall center to individual bin's
        # center, expressed as 0.0 (bin at center) to 1.0 (bin at limit of
        # lower-to-upper range).
        bin_center = log(bin_index + 0.5, 2) / spectrum_bits
        dist = abs(bin_center - mid) / half_width
        if dist < 1.0:
            # Filter out a few math stragglers at either end
            # Bin weights have a cubic falloff curve within range:
            dist = 1.0 - dist  # Invert dist so 1.0 is at center
            bin_weights.append(((3.0 - (dist ** 2.0)) * dist) ** 2.0)
        # Scale bin weights so total is 1.0 for each column, but then mute
        # lower columns slightly and boost higher columns. It graphs better.
        total = sum(bin_weights)
        bin_weights = [(weight / total) * (0.8 + idx / glasses.width * 1.4)
                       for idx, weight in enumerate(bin_weights)]
    # List w/five elements is stored for each column:
    # 0: Index of the first spectrum bin that impacts this column.
    # 1: A list of bin weights, starting from index above, length varies.
    # 2: Color for drawing this column on the LED matrix. The 225 is on
    # purpose, providing hues from red to purple, leaving out magenta.
# 3: Current height of the 'falling dot', updated each frame
# 4: Current velocity of the 'falling dot', updated each frame
column_table.append(
    [
        first_bin - low_bin,
        bin_weights,
        colorwheel(225 * column / glasses.width),
        glasses.height,
        0.0,
    ]
)
# print(column_table)

# MAIN LOOP ***************
dynamic_level = 10  # For responding to changing volume levels
frames, start_time = 0, monotonic()  # For frames-per-second calc
while True:
    # The try/except here is because VERY INFREQUENTLY the I2C bus will
    # encounter an error when accessing the LED driver, whether from bumping
    # around the wires or sometimes an I2C device just gets wedged. To more
    # robustly handle the latter, the code will restart if that happens.
    try:
        mic.record(rec_buf, fft_size)  # Record batch of 16-bit samples
        samples = np.array(rec_buf)  # Convert to ndarray
        # Compute spectrogram and trim results. Only the left half is
        # normally needed (right half is mirrored), but we trim further as
        # only the low bin to high bin elements are interesting to graph.
        spectrum = spectrogram(samples)[low_bin : high_bin + 1]
        # Linearize spectrum output. spectrogram() is always nonnegative,
        # but add a tiny value to change any zeros to nonzero numbers
        # (avoids rare 'inf' error)
        spectrum = np.log(spectrum + 1e-7)
        # Determine minimum & maximum across all spectrum bins, with limits
        lower = max(np.min(spectrum), 4)
        upper = min(max(np.max(spectrum), lower + 6), 20)
        # Adjust dynamic level to current spectrum output, keeps the graph
        # 'lively' as ambient volume changes. Sparkle but don't saturate.
        if upper > dynamic_level:
            # Got louder. Move level up quickly but allow initial "bump."
            dynamic_level = upper * 0.7 + dynamic_level * 0.3
        else:
            # Got quieter. Ease level down, else too many bumps.
            dynamic_level = dynamic_level * 0.5 + lower * 0.5
        # Apply vertical scale to spectrum data. Results may exceed
        # matrix height...that's OK, adds impact!
        data = (spectrum - lower) * (7 / (dynamic_level - lower))
        for column, element in enumerate(column_table):
            # Start BELOW matrix and accumulate bin weights UP, saves math
            first_bin = element[0]
            column_top = glasses.height + 1
            for bin_offset, weight in enumerate(element[1]):
                column_top -= data[first_bin + bin_offset] * weight
            if column_top < element[3]:  # Above current falling dot?
                element[3] = column_top - 0.5  # Move dot up
                element[4] = 0  # and clear out velocity
            else:
                element[4] += 0.2  # and accelerate
            column_top = int(column_top)  # Quantize to pixel space
        for row in range(column_top):  # Erase area above column
            # Erase area above column
glasses.pixel(column, row, 0)
for row in range(column_top, glasses.height):
    glasses.pixel(column, row, element[2])
glasses.pixel(column, int(element[3]), 0xE08080)  # Draw peak dot
glasses.show()  # Buffered mode MUST use show() to refresh matrix
frames += 1
    # print(frames / (monotonic() - start_time), "FPS")
except OSError:
    # See "try" notes above regarding rare I2C errors.
    print("Restarting")
    reload()

Upload the Code and Libraries to the Feather nRF52840 Sense

After downloading the Project Bundle, plug your Feather nRF52840 Sense into the computer's USB port. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called CIRCUITPY. Unzip the folder and copy the following items to the Feather nRF52840 Sense's CIRCUITPY drive.

- lib folder
- code.py

Your Feather nRF52840 Sense CIRCUITPY drive should look like this after copying the lib folder and the code.py file.
3D Printing

CAD Parts List
STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below:

- case.stl
- pcb-mount.stl
- cover.stl

[ cad-files.zip](https://adafruit.it/Vcv)

[ STLs.zip](https://adafruit.it/Vcw)

CAD Assembly
The IS31FL3741 and Feather Sense are secured to the PCB mount using 8x M2.5 x 10mm screws and hex nuts. The PCB mount is secured to the case with M3 x 6mm long screws. The cover snap fits on to the back of the case.
Slicing Parts
No supports are required. Slice with setting for PLA material.

The parts were sliced using CURA using the slice settings below.

- PLA filament 220c extruder
- 0.2 layer height
- 10% gyroid infill
- 60mm/s print speed
- 60c heated bed

Design Source Files
The project assembly was designed in Fusion 360. This can be downloaded in different formats like STEP, STL and more. Electronic components like Adafruit's board, displays, connectors and more can be downloaded from the Adafruit CAD parts GitHub Repo (https://adafruit.it/AW8).

Assembly

Hardware for PCBs
Use the following hardware to secure the LED matrix and Feather Sense to the PCB Mount.

- 8x M2.5 x 10mm long screws
- 8x M2.5 hex nuts
Install LED Matrix
Place the LED matrix over the PCB mounts with matching standoffs. Insert the M2.5 x 10mm long screws through the mounting holes and standoffs.

Secure LED Matrix
Use hex nuts to secure the LED matrix to the PCB mount. Reference the photos for correct orientation.
Install and Secure Feather
Place the Feather Sense over the remaining standoffs on the PCB mount. Line up the mounting holes and insert M2.5 x 10mm long screws through the holes. Use M2.5 hex nuts to secure the Feather to the PCB mount.

Install PCB Mount to Case
Orient the tabs from PCB mount with standoffs in the Case. Use M3 x 6mm screws to secure the PCB mount to the case.
Installed PCB Mount
Place the PCB mount into the case with the mounting tabs lined up with the standoffs.

Secure PCB Mount to Case
Insert and fasten M3 x 6mm long screws through the mounting tabs. The threads of the screw should self tap into the standoffs in the case.

Install Cover
The back cover is symmetrical so it can be installed in either orientation. Place the cover over the case and firmly press together to snap fit shut.
Final Build
Congratulations on building your Mini LED matrix audio visualize!