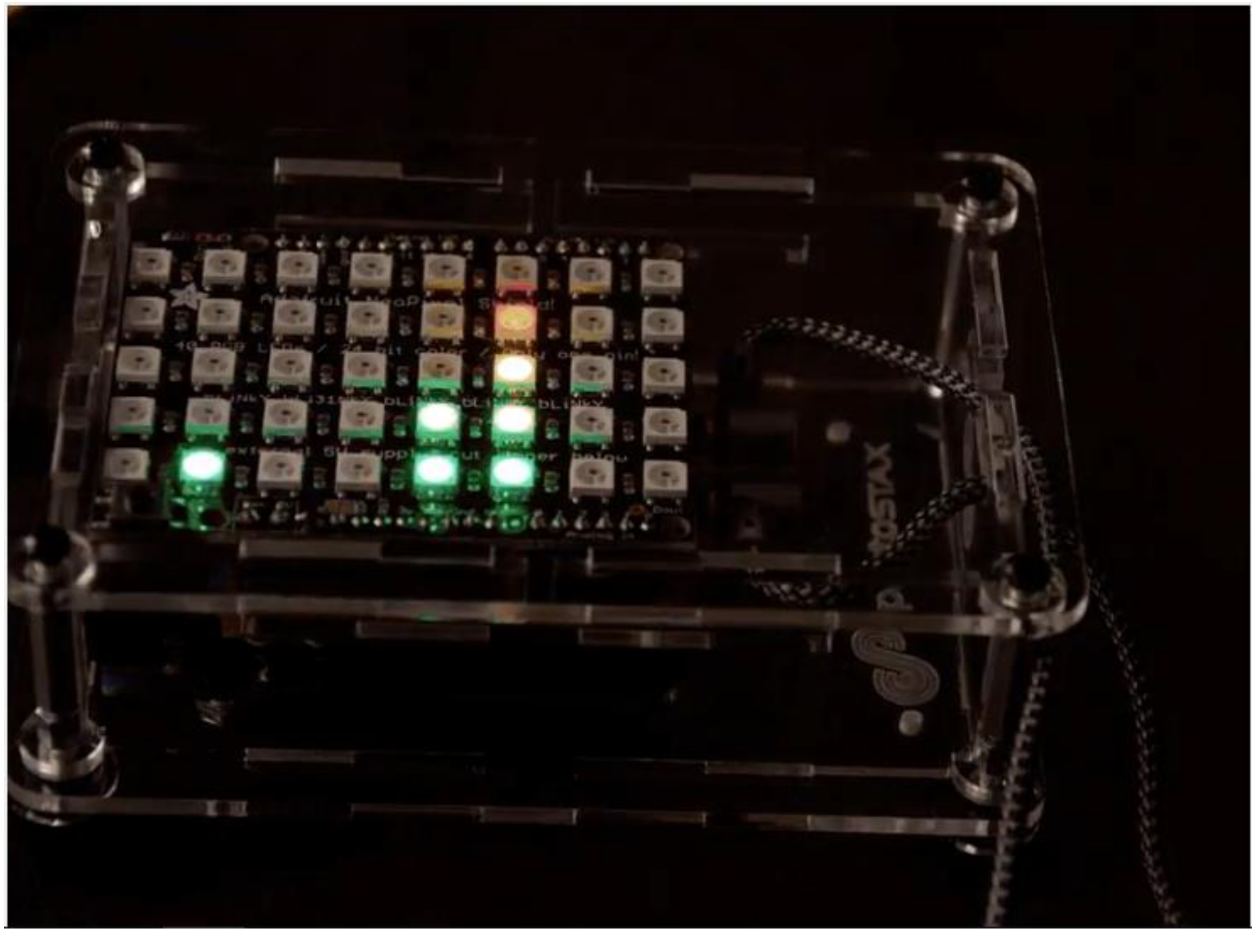
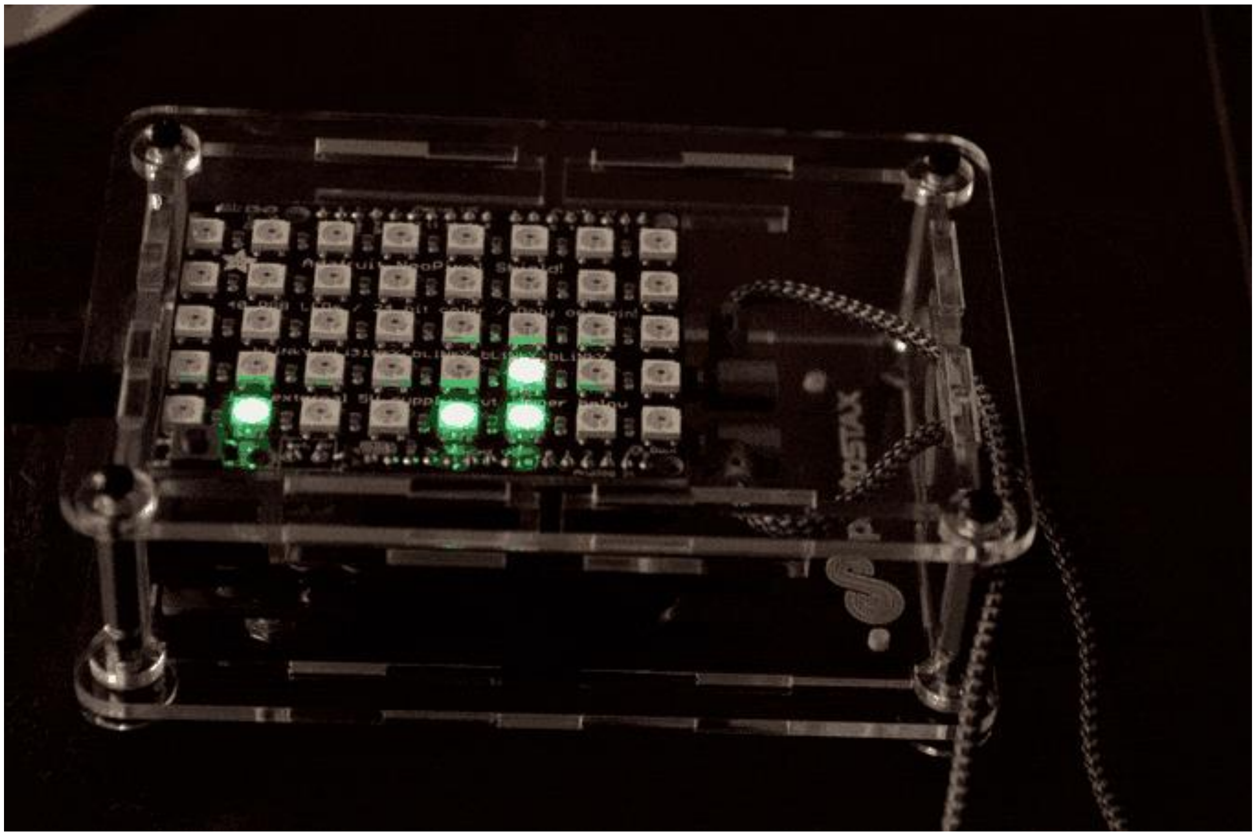


RGB Matrix Audio Visualizer with Arduino

About this project



ProtoStax Audio Visualizer Live Demo _ https://youtu.be/n20gm_qOhbE

In this article, I explain how to build an RGB LED matrix audio visualizer, using an Arduino, an RGB LED Matrix Shield, and an Audio Spectrum Analyzer Shield, then put it in an enclosure so that you can have a finished project that you can display by your music system to have a nice light show of your music!

For the Audio Spectrum Analysis, I use SparkFun's Spectrum Shield, that uses two MSGEQ7 graphic equalizer display filters, that split a stereo audio input into 7-bands (per channel) and read the amplitude of each using the ADC on your Arduino. It comes with Arduino sample sketches to get started.

For the RGB LED Matrix, I use Adafruit's NeoPixel Shield for Arduino, that consists of 40 RGB NeoPixels (Adafruit's terminology for their WS2812 light source). Red, green and blue LEDs are integrated alongside a driver chip into a tiny surface-mount package controlled through a single wire. They can be used individually, chained into longer strings or assembled into still more interesting form-factors. In the case of the Shield, they are chained together. The Shield also comes with the Adafruit_NeoMatrix library, that simplifies access to the RGB LED Matrix and controlling the LEDs.

Lastly comes the enclosure. Some of you may be aware that I've created a new stackable, modular enclosure system called ProtoStax. It was a personal itch that I had to scratch - I wanted an enclosure that supported different stages of prototyping, offering protection and open access when starting out, with the ability to add side walls and the top later, but also have the ability to stack multiple units either side-by-side or one on top of the other, thereby having the ability to expand with prototyping needs and the addition of other boards and components.

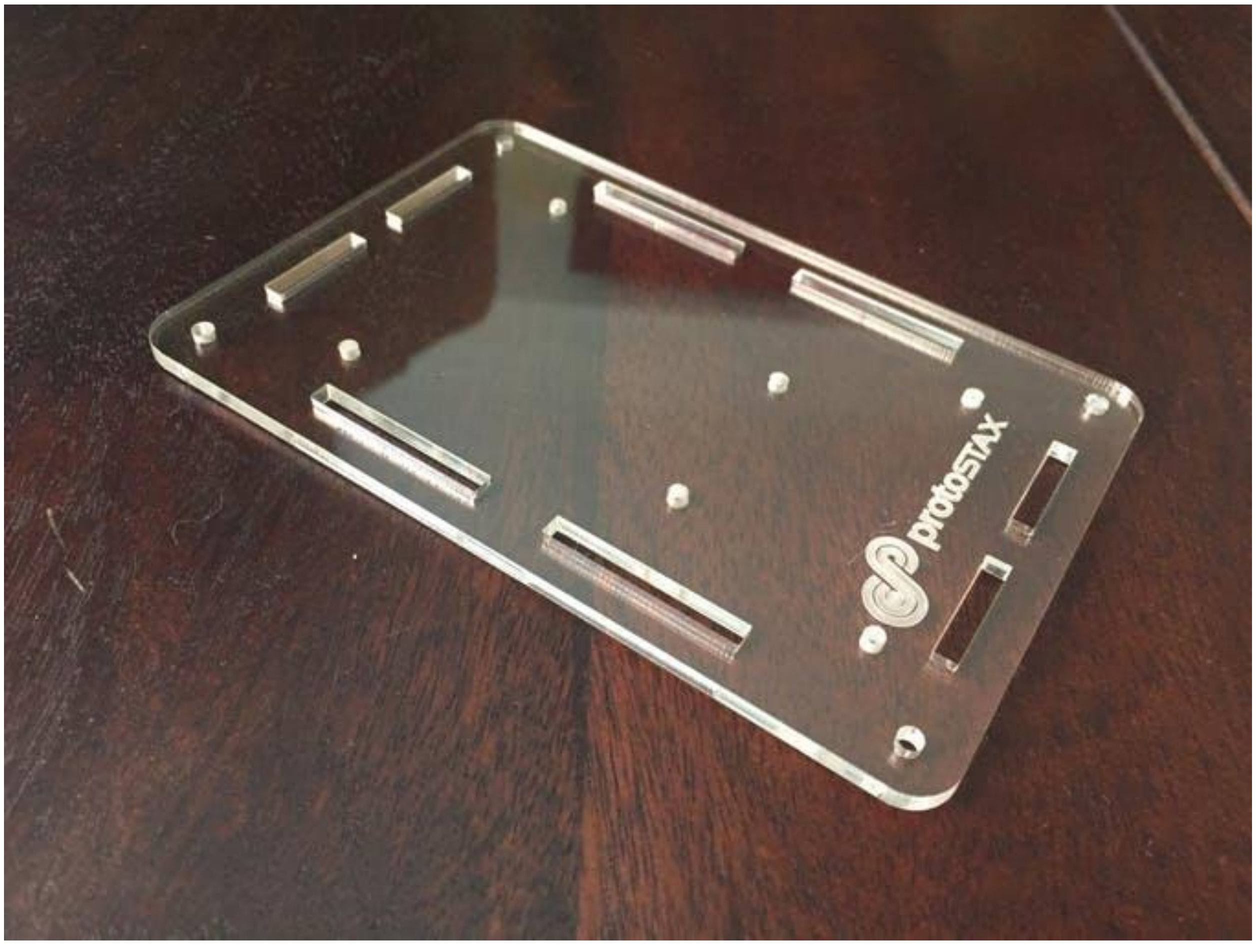
In this example I use ProtoStax for Arduino, a clear acrylic enclosure for the Arduino -- it fits both the Uno/Leonardo footprint as well as the larger Mega/Due footprint -- that is also stackable and modular and has room comfortably for the two Shields (with some minor modifications, that I will outline). It is clear and sturdy and also has rubber feet to slightly elevate it and protect the surface of your table, so you can display your Audio Visualizer and its light show along with your music system! ☺

Ok, let's get started, shall we? ☺

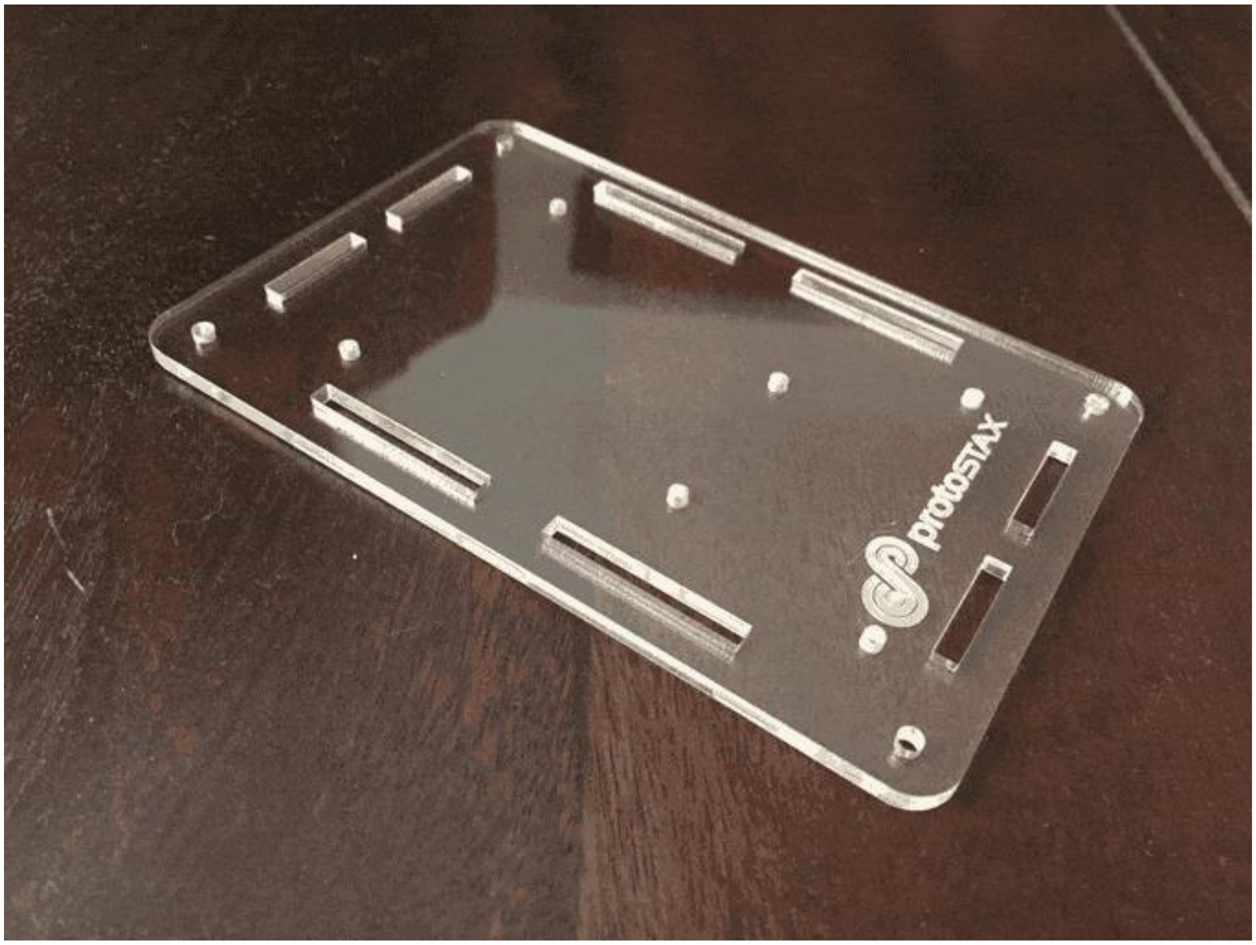
Step 1 - Mount the Arduino to the Enclosure Base Plate

Let us first mount the Arduino (Uno in our example) to the enclosure's base plate. This gives it protection while offering full open access to it to configure and setup the Arduino and play around with it. When you are ready to close it up, It is easy to add the side walls and top plate and secure everything with screws.

Mount the Arduino to the base plate, and add feet and other hardware to prepare the enclosure in *Platform Configuration*. See steps below in the slideshow - the caption for each image is numbered and gives additional explanation for each step.



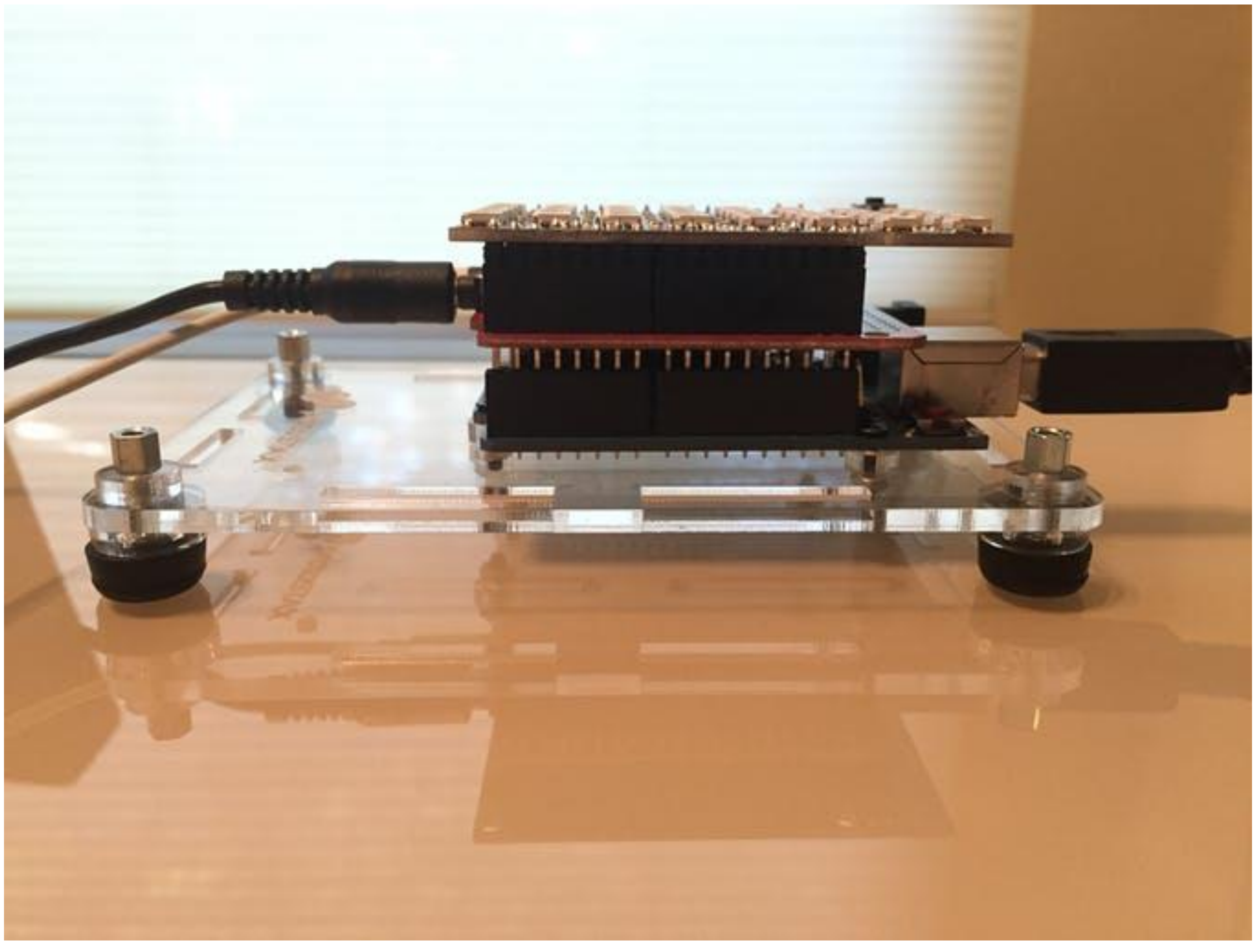
Here are all the steps as an animated gif:



Step 2 - Prepare the SparkFun Spectrum Shield for the Arduino

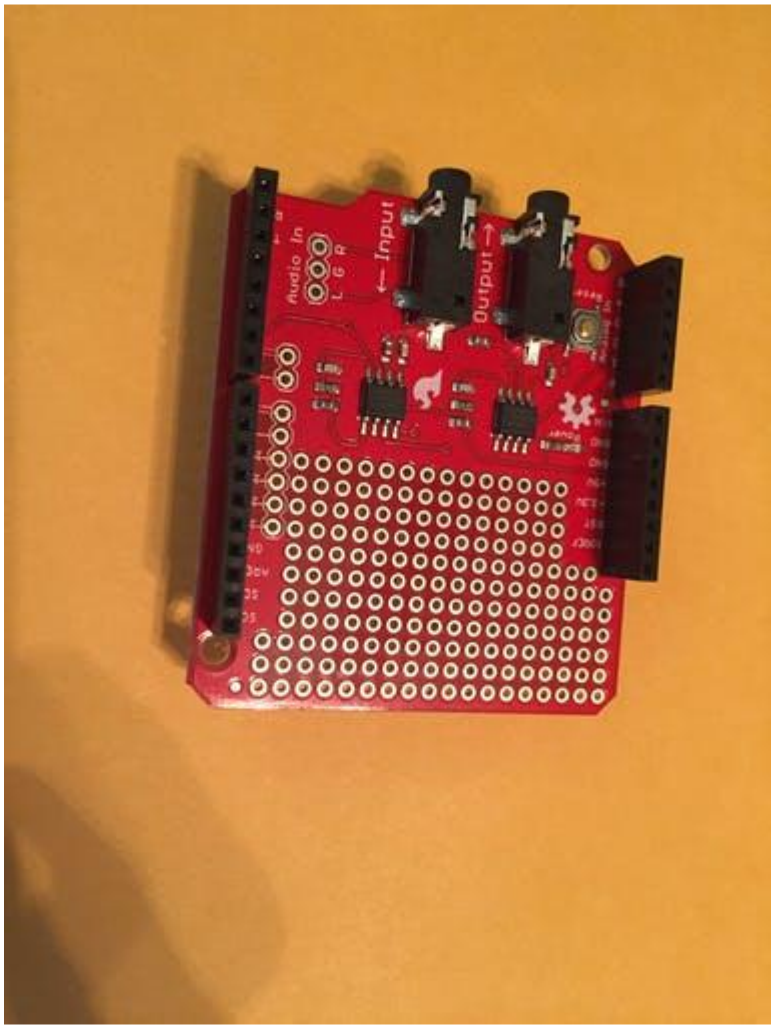
The SparkFun Spectrum Shield does not come with headers. Fortunately for us, Adafruit NeoPixel Shield for Arduino comes with both stacking headers and plain headers. Since I want the NeoPixel Shield to be at the top, I want to use plain headers with it so that it will be flush, and this leaves the stacking headers for use with the Spectrum Shield, which is just what I want! ☺

However, the Spectrum Shield with stacking headers does not fit snugly - the USB and Power ports on the Arduino Uno get in the way, as shown in the picture below.

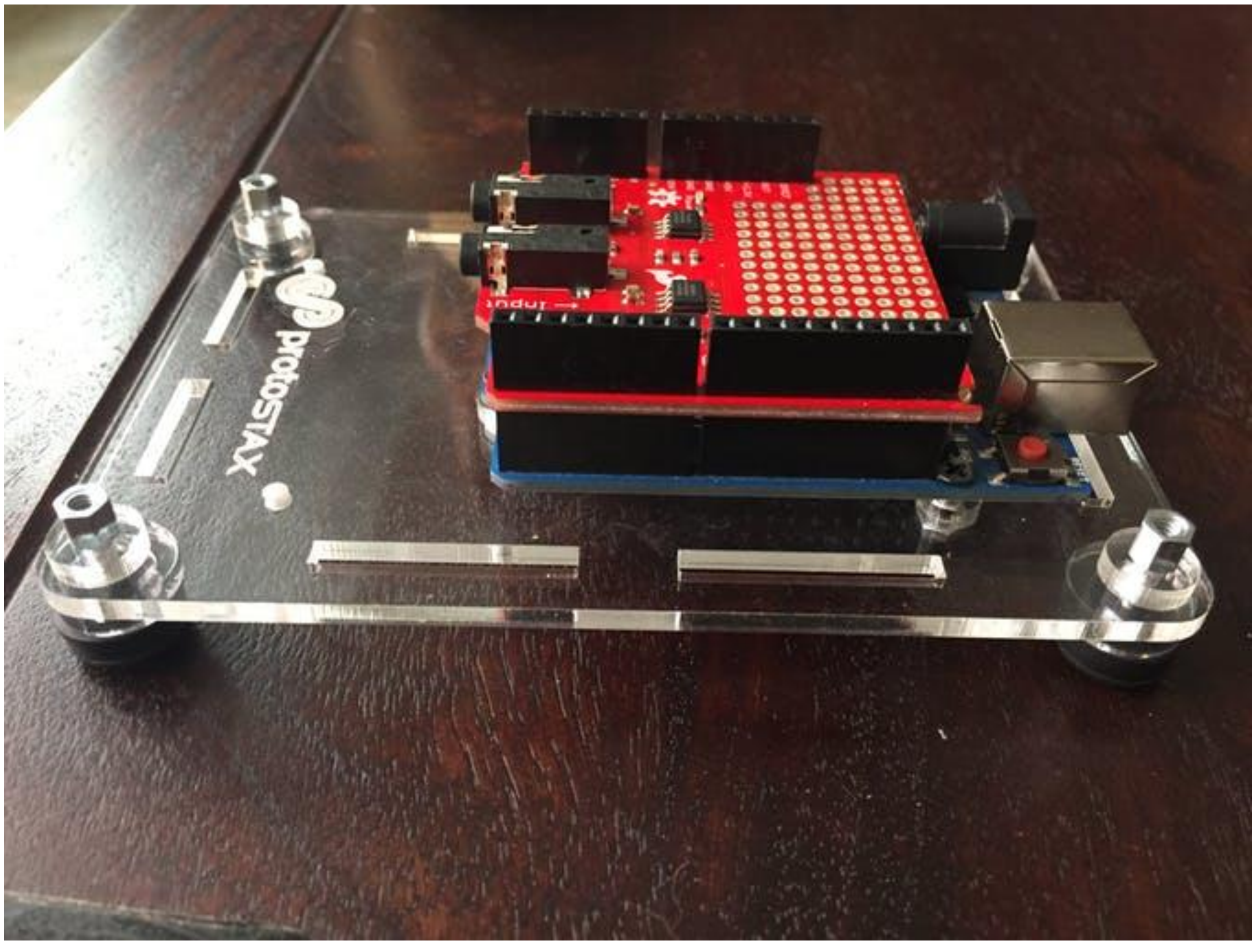


I made the following two modifications -

- Cut off the end of the Spectrum Shield over the USB and Power ports (that part has prototyping area, which is not used. Even if you are using it, you would end up only cutting off one row of holes) This makes the Spectrum Shield sit snugly on the Arduino.
- The legs of the stacking headers might still be too long for the Spectrum Shield to sit snugly. I trimmed off the legs of the stacking headers a hair to make the Spectrum Shield sit snugly on the Arduino with the stacking headers.



Now it fits snugly!

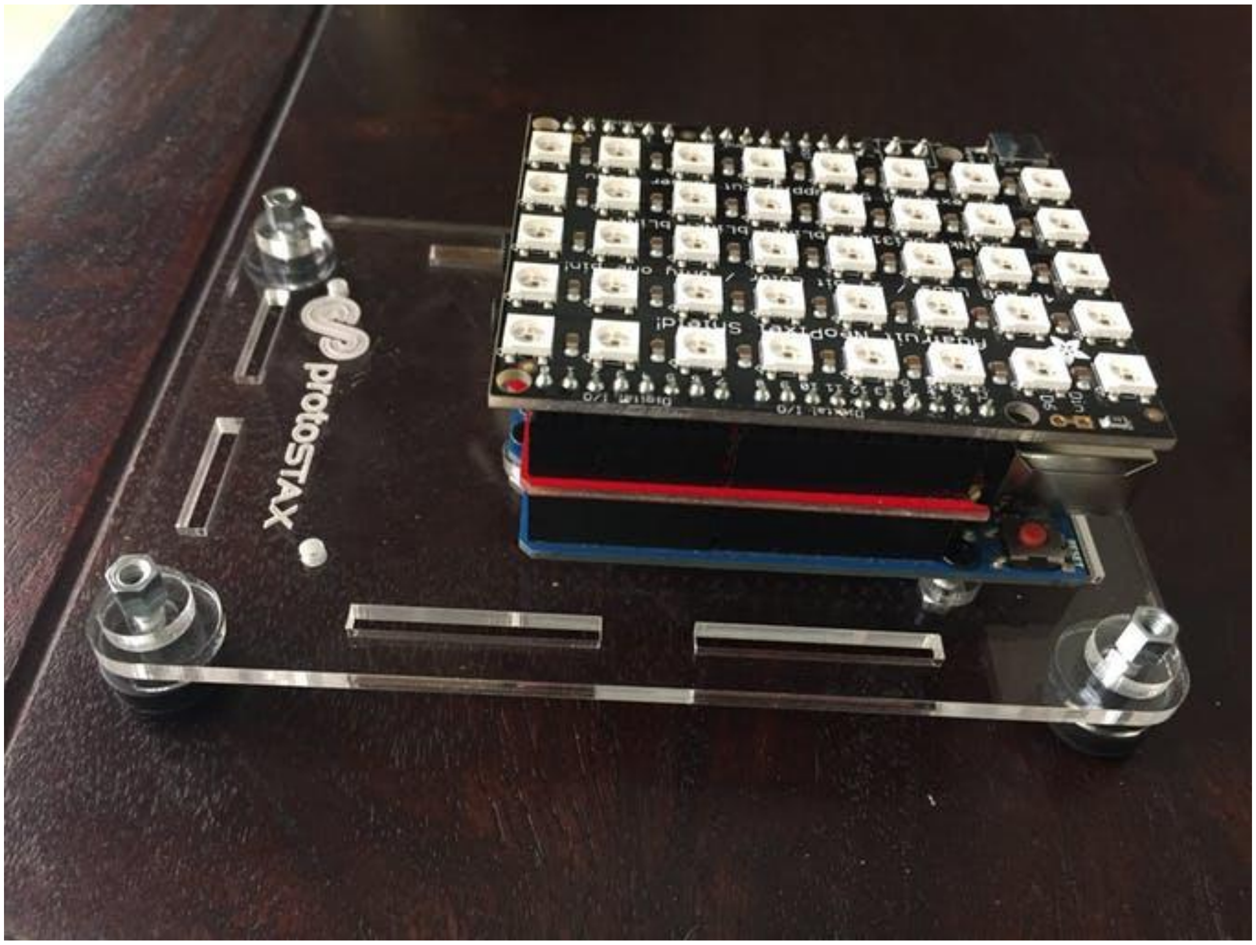


Step 3 - Insert the Adafruit NeoPixel Shield for Arduino into the stacking header of the Spectrum Shield

The Adafruit NeoPixel Shield goes on top of the Spectrum Shield. You will first need to solder in the regular headers (that it came with). I also soldered in the terminal connector that it came with, though in this example, I'm powering it using the Arduino, as all the LEDs will not be turning on simultaneously, so the power consumption is within the amounts that the Arduino can provide.

Message from Adafruit's NeoPixel Shield for Arduino page:

To make it easy to start, the LEDs are by default powered from the 5V onboard Arduino supply. As long as you aren't lighting up all the pixels full power white that should be fine. If you want to power the shield with an external power supply, solder in the included terminal block (pro-tip: put it on the bottom of the board so it doesn't stick up) to wire in an external 4-6VDC power supply - that power supply will also power the Arduino and shield. If you want to use the terminal block to power the shield but keep the Arduino itself on DC or USB power only, cut the center of the solder jumper to the right of the terminal block. There's a polarity protection FET on the external input in case you wire the power backwards (we would never do that, it was, umm, a friend of ours, yeah that's it!)



Step 4 - Demo Code

Let us take a look at the Demo Code and see what it is doing. To do that, we can split it into two main parts:

- Spectrum Analysis and saving results
- Translating that into a display/color scheme for the 8x5 NeoPixel Matrix.

You can quickly reference the demo code here:

https://github.com/protostax/ProtoStax_Audio_Visualizer_Demo/blob/master/ProtoStax_Audio_Visualizer_Demo.ino

Spectrum Analysis

You can refer to the Spectrum Shield Hookup Guide for additional information pertaining to the Spectrum Shield. I've summarized the information here.

By writing a digital sequence to the STROBE and RESET pins of the Spectrum Shield, you initialize the MSGEQ7 chips used by the Shield. You can then proceed to read the magnitude of each of the 7 different frequency bands that the spectrum is split into. Each band is read followed by pulsing of the STROBE pin to initiate reading of the next band. The values are stored in Frequencies_One[7] and Frequencies_Two[7] for the two channels of the stereo input. The values are read by using the 10-bit ADCs of the Arduino, and the output value can thus be 0 - 1023 - they provide a representation of the amplitude of each frequency band.

```
//Declare Spectrum Shield pin connections
#define STROBE 4
#define RESET 5
#define DC_One A0
#define DC_Two A1
//Define spectrum variables
int freq_amp;
int Frequencies_One[7];
int Frequencies_Two[7];
int i;

void setup() {
...
 //Initialize Spectrum Analyzers
 digitalWrite(STROBE, LOW);
 delay(1);
 digitalWrite(RESET, HIGH);
 delay(1);
 digitalWrite(STROBE, HIGH);
 delay(1);
 digitalWrite(STROBE, LOW);
 delay(1);
 digitalWrite(RESET, LOW);
...
}

void loop() {
...
  Read_Frequencies();
...
}

/*******************Pull frquencies from Spectrum Shield********************/
void Read_Frequencies(){
...
 //Read frequencies for each band
 for (freq_amp = 0; freq_amp<7; freq_amp++)
 {
   Frequencies_One[freq_amp] = (analogRead(DC_One) + analogRead(DC_One) ) >>
1 ;
   Frequencies_Two[freq_amp] = (analogRead(DC_Two) + analogRead(DC_Two) ) >>
1;
...
   digitalWrite(STROBE, HIGH);
   digitalWrite(STROBE, LOW);
 }
}
```

The 7 bands of the frequency spectrum are:

- 63Hz
- 160Hz
- 400Hz
- 1kHz
- 2.5kHz
- 6.25kHz
- 16kHZ

I split these up into 3 ranges - BASS, MID_RANGE and TREBLE. Typical Bass range is 60 to 250 Hz, so the first two bands are in the BASS range. Mid-range frequencies are typically 500 Hz to 2 kHz, so I group the next 3 bands into MID_RANGE. I group the remaining 2 bands into TREBLE.

[Note: I also note the max reading of each of the bands into a separate variable. This can possibly be used for automatically scaling the readings to the level represented by the RGB matrix columns - this is useful in the case where the input signal is low - otherwise only very few of the RGB matrix would light up in that case.]

RGB Matrix

You can refer to the Adafruit NeoPixel Überguide for additional information pertaining the NeoPixel Shield and NeoPixels in general. I've summarized the information pertaining to our use here.

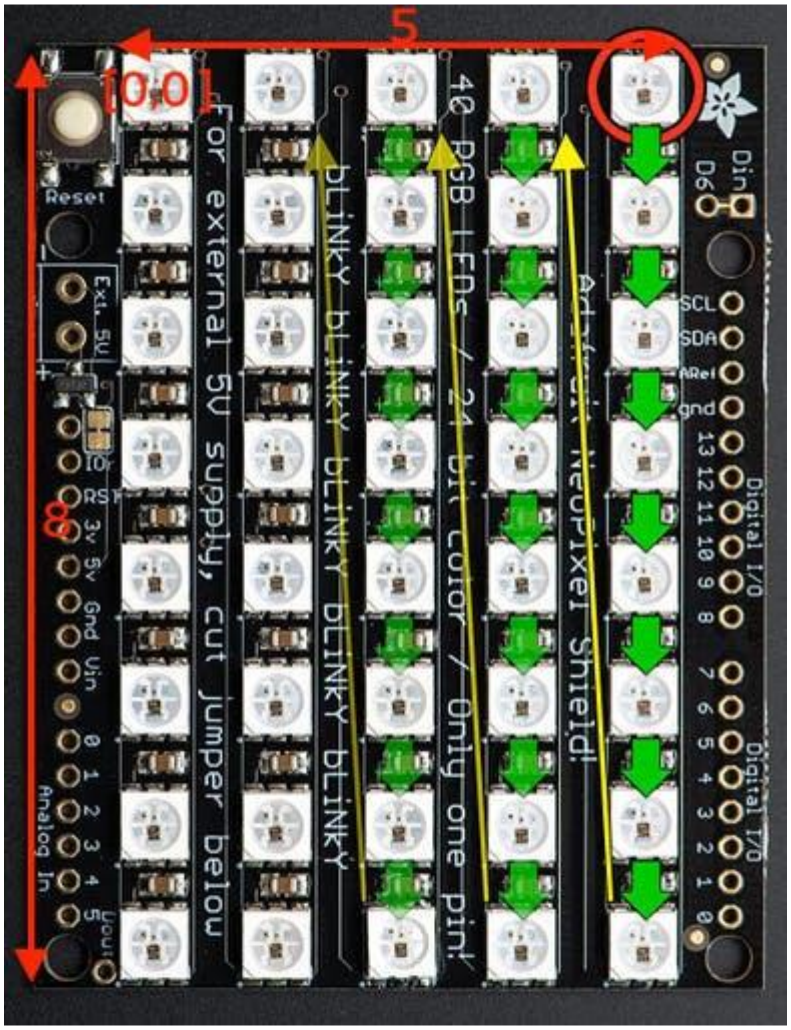
The main point I will try to clarify, which I found a little confusing at first, is the direction and orientation of the NeoPixel Shield and the numbering of the coordinate system. The Überguide explains it, but I think I can make it a little easier.

First thing to note is that in the coordinate system, [0, 0] ALWAYS refers to the top left, regardless of the orientation.

Next is to note the WIDTH, followed by HEIGHT, of whatever orientation you are interested in (i.e. 5 x 8 vs 8 x 5 in the case of our Shield)

Third is to note the position of the PHYSICAL LED #0 (marked by the cute Adafruit logo). TOP-RIGHT, TOP-LEFT, BOTTOM-LEFT and BOTTOM-RIGHT as the case may be. Also note the orientation of the progression of the physical LEDS. The layout is PROGRESSIVE in our board (the next physical led after the end of one row starts at the beginning of the next row as indicated by the YELLOW line). The orientation of the progression is along the ROWS for when the width is broader (horizontal orientation) (as indicated by the short GREEN arrows), and COLUMNS with the width is narrower (vertical orientation) (again, as indicated by the short GREEN arrows).

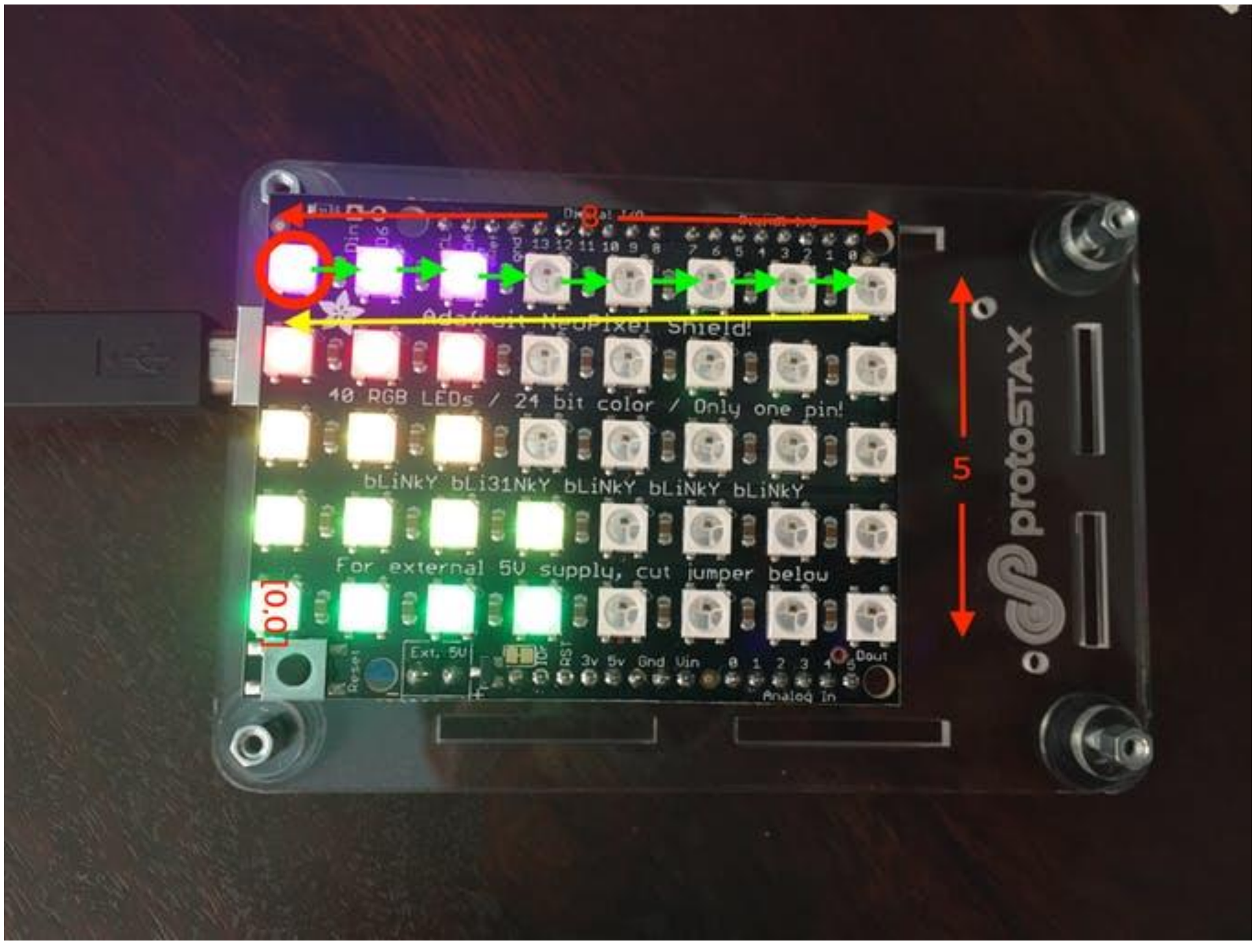
These are illustrated by the 4 images below in the slideshow. The captions include the pertinent settings in each case!


5
[0,0]
8
40 RGB LEDs / 24 bit Color / Only one pin!
Adafruit NeoPixel Shield!
For external 5V supply, cut jumper below
bLiNKY bLiNKY bLiNKY bLiNKY
Reset
Din
D6

In our example, we have 7 bands of frequency, and an 8 x 5 (or 5 x 8, depending on which way you look at it!) matrix. I chose to display the 7 bands along the 8 dimension (leaving one unused). I would then display the amplitude representation of each frequency band along the 5 dimension. In other words, I want my progression to be as follows:

https://youtu.be/0mGiqoVA6K4

I want my origin to start at the lower left corner (representing the lowest level of the lowest frequency band) and work its way upwards. But since the first thing to note in the coordinate system is that [0, 0] always refers to TOP-LEFT, you should tilt your head to the left and look at the image below, to understand the choice of values for initializing the NeoMatrix! ☺(WIDTH = 5, HEIGHT = 8, TOP-RIGHT, COLUMNS PROGRESSIVE)


8
5
[0,0]
Adafruit NeoPixel Shield!
40 RGB LEDs / 24 bit color / Only one pin!
bLiNkY bLi31NkY bLiNkY bLiNkY bLiNkY
For external 5V supply, cut jumper below
protoSTAX

Let's delve a little in the demo code pertaining to the NeoMatrix and graphing the frequencies. Firstly, we have determined that our NeoPixel has WIDTH=5, HEIGHT=8, and the orientation we like is TOP-RIGHT, and COLUMNS PROGRESSIVE. Follow the setup required for the matrix in the setup() function.

In the loop(), we read any Serial input to select the color scheme - I've defined 3 different color schemes

```
enum SCHEME {
 MAGNITUDE_HUE = 0,
 MAGNITUDE_HUE_2 = 1,
 HSV_COLOR_WHEEL = 2
};
```

I then call Graph_Frequencies with that color scheme choice. Note also the first parameter that can select the range of frequencies to display (BASS, MID-RANGE or TREBLE)

```
enum RANGE {
 BASS = 0,
 MID_RANGE = 1,
 TREBLE = 2,
 ALL = 3
};
```

For now, I'm selecting all the ranges to show - it is left as an exercise to the reader to implement selection of ranges to display - either via Serial input or by including a momentary press button to toggle the display between BASS, MID_RANGE, TREBLE or ALL. The selection of the RANGE determines the "from" and "to" range of the rows to be displayed.

For each row (frequency band), we pick the larger of the two frequency magnitudes (right channel and left channel of the stereo input). That value lies between 0 and 1023 as we have already discussed. We need to map that into 5 distinct columns of the display, so we divide the frequency by the FREQ_DIV_FACTOR which is defined as 204 (1023/204 = 5, which will map an output of 1023 to 5). Just to be safe, we also make sure the numCol to display is not larger than 5. This determines the number of columns to display for each frequency band.

I then use matrix.drawPixel() to display the appropriate pixel at the appropriate color.

I use the HSV color wheel in my graphical display. This posed some extra wrinkles to overcome.

Typically, the usage is matrix.drawPixel(column, row, Color(r, g, b)), where Color(r, g, b) represents a color as specified by RED, GREEN and BLUE values. However, using HSV allows for some nice smooth color transitions.

NeoMatrix provides the matrix.ColorHSV(uint16_t hue) method that takes a single uint16_t hue value and returns a uint32_t HSV color.

However matrix.Color(r, g, b) returns a uint16_t color. matrix.drawPixel also expects a 16 bit color.

The way around this is to use matrix.setPassThruColor(32 bit color value). This sets a flag in matrix that causes drawPixel to ignore its color argument and instead use the 32 bit color already set by the above method. Just remember to call matrix.setPassThruColor() to reset the flag in question. Not super elegant, but it works. For example,

```
  static uint16_t hue = 0; //21845 22250 to -250
  uint16_t hueDelta = 200;
  hue += hueDelta;
...
       rgbcolor = matrix.ColorHSV(hue);
...
       matrix.setPassThruColor(rgbcolor);
       matrix.drawPixel(col, row, (uint16_t)0); // color does not matter here
       matrix.setPassThruColor();
...
matrix.show();
```

With HSV, it is possible to increment the 16 bit hue and generate the HSV color code, thereby getting nice smooth transitions in color.

Here are the different code pieces for reference:

```
#define NEO_MATRIX_WIDTH 5
#define NEO_MATRIX_HEIGHT 8
#define NEOPIXEL_PIN 6 // Shield maps it to pin 6
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(NEO_MATRIX_WIDTH,
NEO_MATRIX_HEIGHT, NEOPIXEL_PIN,
 NEO_MATRIX_TOP     + NEO_MATRIX_RIGHT +
 NEO_MATRIX_COLUMNS + NEO_MATRIX_PROGRESSIVE,
 NEO_GRB            + NEO_KHZ800);

....

void setup() {
...
 matrix.begin();
 matrix.setTextWrap(false);
 matrix.setBrightness(40);
 matrix.fillScreen(0);
 matrix.show();
...
}

void loop() {
 static int scheme = 0;
 while (Serial.available() > 0) {
   scheme = Serial.parseInt();
 }
...
 Graph_Frequencies(ALL, scheme);
...
 delay(50);
}

void Graph_Frequencies(CHANNEL c, SCHEME s){
...
  for( row= from; row<to; row++)
  {
    int freq = (Frequencies_Two[row] >
Frequencies_One[row])?Frequencies_Two[row]:Frequencies_One[row];
    int numCol = (freq/FREQ_DIV_FACTOR);
    if (numCol > 5) numCol = 5;
    for (int col = 0 ; col < numCol ; col++) {
... // pick color scheme to display
       matrix.setPassThruColor(rgbcolor);
       matrix.drawPixel(col, row, (uint16_t)0); // color does not matter here
       matrix.setPassThruColor();
       //matrix.show();
    }
    matrix.show();
  }
}
```

Next is the color scheme selection. Note that I have made provisions to be able to select colors for different frequency ranges (bassHue, midHue, trebleHue). I have created 3 different color schemes - one that uses green to red/pink range for display from lowest amplitude to highest, and the other that uses a more pink/blue shifted range. The 3rd scheme uses the same color for all pixels, but cycles through the whole color wheel as it goes along. I'll show you a video of all 3 color schemes.

```
switch(s) {
 case MAGNITUDE_HUE:
   bassHue = 22250;
   midHue = 22250; //54613
   trebleHue = 22250; //43690
   if (row >= 0 && row < 2) {
     rgbcolor = matrix.ColorHSV(bassHue - (7416 * col) );
   } else if (row >= 2 && row < 5) {
     rgbcolor = matrix.ColorHSV(midHue - (7416 * col) );
   } else if (row >= 5 && row < 7) {
     rgbcolor = matrix.ColorHSV(trebleHue - (7416 * col) );
   }
   break;
 case MAGNITUDE_HUE_2:
   bassHue = 54613;
   midHue = 54613; //54613
   trebleHue = 54613; //43690
   if (row >= 0 && row < 2) {
     rgbcolor = matrix.ColorHSV(bassHue - (7416 * col) );
   } else if (row >= 2 && row < 5) {
     rgbcolor = matrix.ColorHSV(midHue - (7416 * col) );
   } else if (row >= 5 && row < 7) {
     rgbcolor = matrix.ColorHSV(trebleHue - (7416 * col) );
   }
   break;
 case HSV_COLOR_WHEEL:
   rgbcolor = matrix.ColorHSV(hue);
   break;
}
```

Step 6 - Testing Everything Out

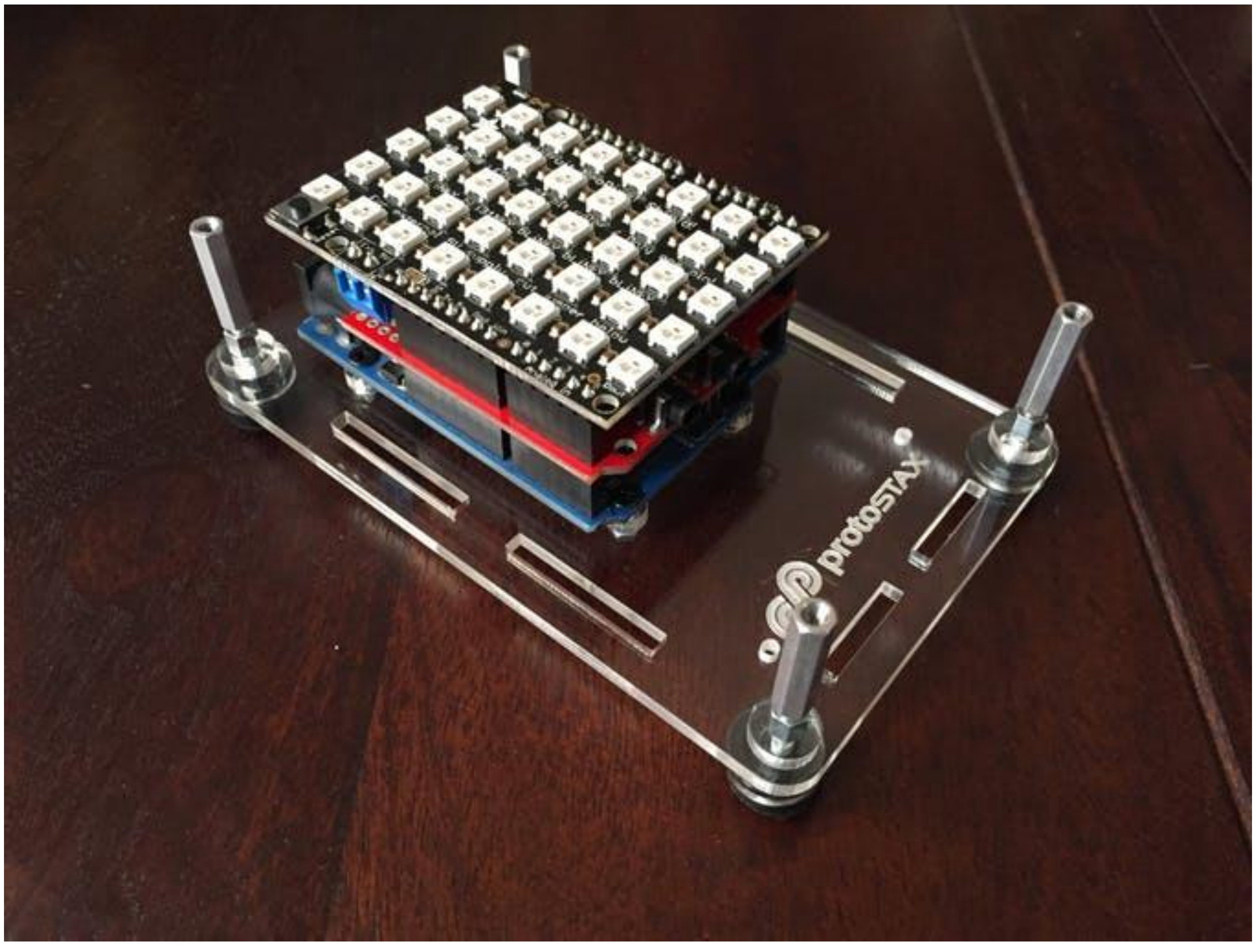
Here is a video of testing everything out

Audio Visualizer Demo Testing _ https://youtu.be/C_OtEn-lWCw

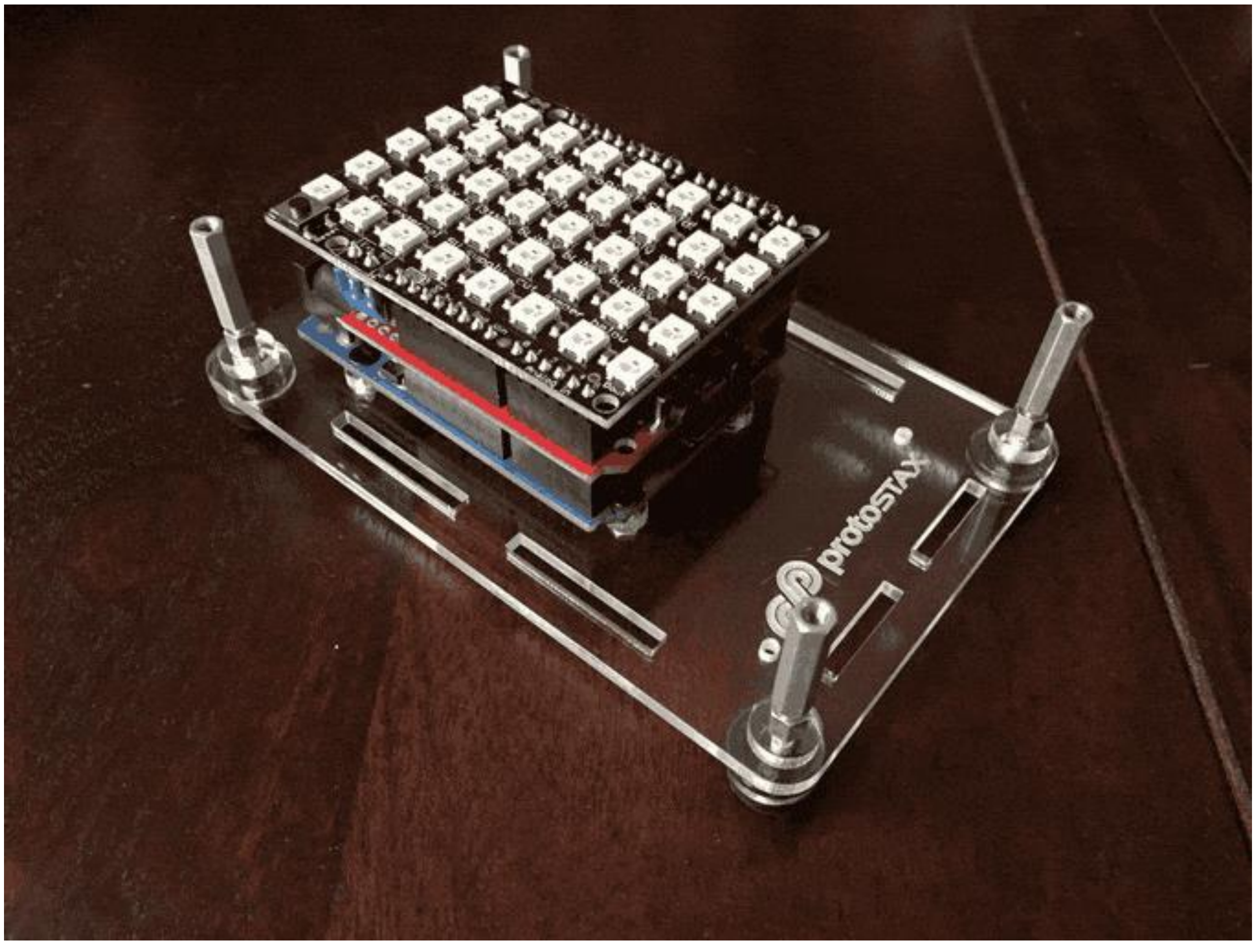
Audio Visualizer Testing showing different color schemes _ https://youtu.be/_VqvKHfRUq0

Step 6 - Close It Up

Install the top bracing elements, side walls, audio cables and top. See steps below in the slideshow - the caption for each image is numbered and gives additional explanation for each step.



Here are all the steps in a single animated gif:



Step 7 - Display Your Creation and Make Beautiful Music Even More Beautiful!

You can now have a fun audio visualizer that you can display alongside your music system and have some cool homemade light action augment your music!

Finished Audio Visualizer. See the different color schemes at play! _ https://youtu.be/n20gm_qOhbE

Step 8 - Taking it further

Here are some ideas to take the project even further!

1. Currently, the audio cables (input and output) are plugged into the Spectrum Shield in the enclosure, and thereby you have this enclosures with these wires connected and dangling outside. Instead, you can add a couple of panel mount stereo jacks (listed in the list of "things" used) to the side wall near the stereo jacks of the Spectrum Shield, and then solder an audio cable with a 3.5mm male stereo audio jack to each, and then plug those instead to the audio jacks of the Spectrum Shield. In this case, your enclosure becomes very neat and all the wiring is self-contained with only audio jacks on the enclosure for external systems to plug into.

2. You can add more even more light schemes to your Audio Visualizer - different color ranges, different patterns.

3. Add the option to enable/disable frequency ranges using the Serial input - currently only color schemes can be changed but not the frequency ranges to display.

4. Add a switch to toggle between different color schemes instead of using Serial input. Modify the enclosure to add a hole to one of the long Side Walls to accommodate a panel mount momentary push button switch (listed in the list of "things" used).

5. Add a second switch to toggle between different frequency ranges displayed (BASS, MID_RANGE, TREBLE, ALL) and mount that switch to the enclosure's side wall.

6. Because the enclosure is made of acrylic, you can use some blue painter's tape over it to protect the surface, and a drill to drill a hole of the requisite side to mount the panel mount stereo jacks and/or switches. It is recommended to use a step drill, or start with a smaller hole and then expand the hole until it is of the size you desire. The stereo jacks listed need a mounting hole of 5/16", and the Switches need a mounting hole of 0.47".

7. Sand the top surface of the top plate lightly and uniformly. This will act as a light diffuser and give you a more diffused and gentler light effect.

If there is sufficient interest, I will update the project to show it with the Audio Jacks and Switches and an Opaque light diffuser top - please send me a message if you would like to see the updated project! ☺

Can you think of any more? Write a comment below to let us know! ☺ Feel free to also ask any questions you may have! ☺

Happy making! ☺

Code

https://github.com/protostax/ProtoStax_Audio_Visualizer_Demo/blob/master/ProtoStax_Audio_Visualizer_Demo.ino

```
/*************************************************

ProtoStax Audio Visualizer Demo



This is a example sketch for an Audio Visualizer using Arduino,
SparkFun Spectrum Shield -->
https://www.sparkfun.com/products/13116 ,

Adafruit NeoPixel Shield -->
https://www.adafruit.com/product/1430 ,

and

ProtoStax for Arduino -->
https://www.protostax.com/products/protostax-for-arduino



It analyzes the frequency spectrum of the audio input and
visualizes it using

an RGB LED matrix with different schemes.



Written by Sridhar Rajagopal for ProtoStax

BSD license. All text above must be included in any
redistribution

*/




#include <Adafruit_GFX.h>

#include <Adafruit_NeoMatrix.h>

#include <Adafruit_NeoPixel.h>
```

```
#ifndef PSTR

#define PSTR // Make Arduino Due happy

#endif


//Declare Spectrum Shield pin connections

#define STROBE 4

#define RESET 5

#define DC_One A0

#define DC_Two A1


//Define spectrum variables

int freq_amp;

int Frequencies_One[7];

int Frequencies_Two[7];


// MATRIX DECLARATION:

// Parameter 1 = width of NeoPixel matrix

// Parameter 2 = height of matrix
// Parameter 3 = pin number (most are valid)

// Parameter 4 = matrix layout flags, add together as needed:

// NEO_MATRIX_TOP, NEO_MATRIX_BOTTOM, NEO_MATRIX_LEFT,
NEO_MATRIX_RIGHT:

// Position of the FIRST LED in the matrix; pick two, e.g.

// NEO_MATRIX_TOP + NEO_MATRIX_LEFT for the top-left corner.

// NEO_MATRIX_ROWS, NEO_MATRIX_COLUMNS: LEDs are arranged in
horizontal

// rows or in vertical columns, respectively; pick one or the
other.

// NEO_MATRIX_PROGRESSIVE, NEO_MATRIX_ZIGZAG: all rows/columns
proceed

// in the same order, or alternate lines reverse direction; pick
one.

// See example below for these values in action.

// Parameter 5 = pixel type flags, add together as needed:

// NEO_KHZ800 800 KHz bitstream (most NeoPixel products w/WS2812
LEDs)

// NEO_KHZ400 400 KHz (classic 'v1' (not v2) FLORA pixels,
WS2811 drivers)

// NEO_GRB Pixels are wired for GRB bitstream (most NeoPixel
products)

// NEO_RGB Pixels are wired for RGB bitstream (v1 FLORA pixels,
not v2)
```

```
// Example for NeoPixel Shield used in Audio Visualizer Demo.

// In this application we'd like to use it

// as a 5x8 tall matrix, with the USB port positioned at the top
of the

// Arduino. When held that way, the first physical pixel is at
the top right, and

// lines are arranged in columns, progressive order. The shield
uses

// 800 KHz (v2) pixels that expect GRB color data.

// See https://www.hackster.io/sridhar-rajagopal/rgb-matrix-
audio-visualizer-with-arduino-845062

// for more details on how to choose the parameters




#define NEO_MATRIX_WIDTH 5

#define NEO_MATRIX_HEIGHT 8



#define NEOPIXEL_PIN 6 // Shield maps it to pin 6
```

```
Adafruit_NeoMatrix matrix = Adafruit_NeoMatrix(NEO_MATRIX_WIDTH,
NEO_MATRIX_HEIGHT, NEOPIXEL_PIN,

NEO_MATRIX_TOP + NEO_MATRIX_RIGHT +

NEO_MATRIX_COLUMNS + NEO_MATRIX_PROGRESSIVE,

NEO_GRB + NEO_KHZ800);



enum RANGE {

BASS = 0,

MID_RANGE = 1,

TREBLE = 2,

ALL = 3

};



enum SCHEME {

MAGNITUDE_HUE = 0,

MAGNITUDE_HUE_2 = 1,

HSV_COLOR_WHEEL = 2

};
```

```
/*******************Setup ************************/

void setup() {

Serial.begin(9600);

Serial.println("ProtoStax Audio Visualizer Demo");

Serial.println("*************************************************
**");



matrix.begin();

matrix.setTextWrap(false);

matrix.setBrightness(40);

matrix.fillScreen(0);

matrix.show();



//Set spectrum Shield pin configurations

pinMode(STROBE, OUTPUT);

pinMode(RESET, OUTPUT);

pinMode(DC_One, INPUT);

pinMode(DC_Two, INPUT);

digitalWrite(STROBE, HIGH);
digitalWrite(RESET, HIGH);


//Initialize Spectrum Analyzers

digitalWrite(STROBE, LOW);

delay(1);

digitalWrite(RESET, HIGH);

delay(1);

digitalWrite(STROBE, HIGH);

delay(1);

digitalWrite(STROBE, LOW);

delay(1);

digitalWrite(RESET, LOW);

}



/************************** Loop*****************************/

void loop() {

static int scheme = 0;

while (Serial.available() > 0) {
```

```
scheme = Serial.parseInt();

}



Read_Frequencies();

Graph_Frequencies(ALL, scheme);

// Print_Frequencies();

delay(50);



}



int max_bass_freq = 0;

int max_mid_freq = 0;

int max_treble_freq = 0;



/******************Pull frquencies from Spectrum
Shield********************/

void Read_Frequencies(){

max_bass_freq = 0;

max_mid_freq = 0;
```

```
max_treble_freq = 0;



//Read frequencies for each band

for (freq_amp = 0; freq_amp<7; freq_amp++)

{

Frequencies_One[freq_amp] = (analogRead(DC_One) +
analogRead(DC_One) ) >> 1 ;

Frequencies_Two[freq_amp] = (analogRead(DC_Two) +
analogRead(DC_Two) ) >> 1;



if (freq_amp >= 0 && freq_amp < 2) {

if (Frequencies_One[freq_amp] > max_bass_freq)

max_bass_freq = Frequencies_One[freq_amp];

if (Frequencies_Two[freq_amp] > max_bass_freq)

max_bass_freq = Frequencies_Two[freq_amp];

}

else if (freq_amp >= 2 && freq_amp < 5) {

if (Frequencies_One[freq_amp] > max_mid_freq)

max_mid_freq = Frequencies_One[freq_amp];

if (Frequencies_Two[freq_amp] > max_mid_freq)
max_mid_freq = Frequencies_Two[freq_amp];

}

else if (freq_amp >= 5 && freq_amp < 7) {

if (Frequencies_One[freq_amp] > max_treble_freq)

max_treble_freq = Frequencies_One[freq_amp];

if (Frequencies_Two[freq_amp] > max_treble_freq)

max_treble_freq = Frequencies_Two[freq_amp];

}



digitalWrite(STROBE, HIGH);

digitalWrite(STROBE, LOW);

}

}



int FREQ_DIV_FACTOR = 204;



/*******************Light LEDs based on
frequencies*****************************/

void Graph_Frequencies(RANGE r, SCHEME s){
```

```
int from = 0;

int to = 0;



switch(r) {

case BASS:

from = 0;

to = 2;

break;

case MID_RANGE:

from = 2;

to = 5;

break;

case TREBLE:

from = 5;

to = 7;

break;

case ALL:

from = 0;

to = 7;
break;

default:

break;

}



// Serial.print("max freq is "); Serial.println(max_freq);

// FREQ_DIV_FACTOR = max_bass_freq/4;

// Serial.print("FREQ_DIV_FACTOR is ");
Serial.println(FREQ_DIV_FACTOR);



static uint16_t hue = 0; //21845 22250 to -250

uint16_t hueDelta = 200;

hue += hueDelta;




uint16_t bassHue = 22250;

uint16_t midHue = 22250; //54613

uint16_t trebleHue = 22250; //43690
```

```
matrix.fillScreen(0);

uint32_t rgbcolor;

for(int row= from; row<to; row++)

{



int freq = (Frequencies_Two[row] >
Frequencies_One[row])?Frequencies_Two[row]:Frequencies_One[row];




int numCol = (freq/FREQ_DIV_FACTOR);

if (numCol > 5) numCol = 5;



for (int col = 0 ; col < numCol ; col++) {

switch(s) {

case MAGNITUDE_HUE:

bassHue = 22250;

midHue = 22250; //54613

trebleHue = 22250; //43690

if (row >= 0 && row < 2) {
```

```
rgbcolor = matrix.ColorHSV(bassHue - (7416 * col) );

} else if (row >= 2 && row < 5) {

rgbcolor = matrix.ColorHSV(midHue - (7416 * col) );



} else if (row >= 5 && row < 7) {

rgbcolor = matrix.ColorHSV(trebleHue - (7416 * col) );

}

break;

case MAGNITUDE_HUE_2:

bassHue = 54613;

midHue = 54613; //54613

trebleHue = 54613; //43690

if (row >= 0 && row < 2) {

rgbcolor = matrix.ColorHSV(bassHue - (7416 * col) );

} else if (row >= 2 && row < 5) {

rgbcolor = matrix.ColorHSV(midHue - (7416 * col) );



} else if (row >= 5 && row < 7) {
```

```
rgbcolor = matrix.ColorHSV(trebleHue - (7416 * col) );

}

break;

case HSV_COLOR_WHEEL:

rgbcolor = matrix.ColorHSV(hue);

break;

}




matrix.setPassThruColor(rgbcolor);

matrix.drawPixel(col, row, (uint16_t)0); // color does not
matter here

matrix.setPassThruColor();



//matrix.show();



}

matrix.show();

}

}
```

```
// Used for debugging

void Print_Frequencies() {

for (int i = 0; i < 7; i++) {

Serial.print("FreqOne["); Serial.print(i); Serial.print( "]:");
Serial.println((Frequencies_One[i]/FREQ_DIV_FACTOR)%5);

Serial.print("FreqTwo["); Serial.print(i); Serial.print( "]:");
Serial.println((Frequencies_Two[i]/FREQ_DIV_FACTOR)%5);



}

}
```