Alexander Buchholz  
Digital Signal Processing  
Project Final Report

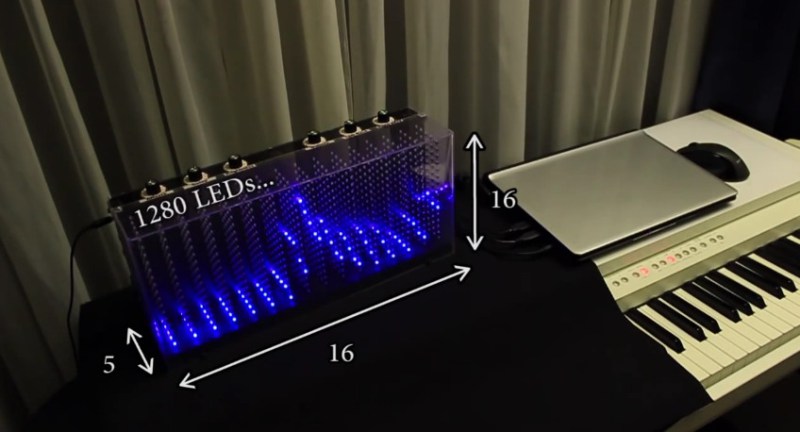
**Audio Spectrum Analyzer Using LEDs**

**Overview**

This project aims to build an audio spectrum analyzer using some colored LEDs and an Arduino Uno. Spectrum analyzers are desired for both professional and recreational use, and are a solid example Digital Signal Processing and the Fast Fourier Transform (FFT). The goal of the project is to show how FFT and clever use of other DSP tools can create interesting products, especially when music is involved.

Spectrum analyzers can be used whenever signals need to be visualized, but this analyzer is built to visualize music and provide an interesting display for recreational use.  
  
**Related Work**

The analyzer is intended to be relatively simple and inexpensive, but the techniques used here can be up-scaled for very impressive results. Figure 1 shows an example of an LED spectrum analyzer scaled into three dimensions. The creator of this analyzer used a DSP technique very similar to the Fourier Transform called Constant Q Transform, which outputs a frequency spectrum on a logarithmic scale─ perfect for audio applications[1]. Volume Unit meters (VU meters) are also used to give a better visual of the spectrum. These tools require more expensive equipment to be utilized effectively, and are above the scope of this class. Still, projects like these capture the interest of professionals, hobbyists, and consumers alike.

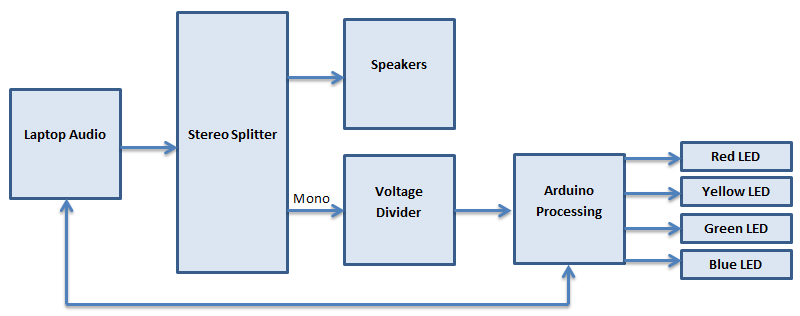


**Figure 1- 3D LED Spectrum Analyzer**

**Data Collection**

For this analyzer, the signal is sampled from a laptop’s headphone jack. Due to the configuration of the Arduino Uno, the maximum sampling rate possible is somewhere between 9000 Hz and 9600Hz. This means the system can cover a Nyquist frequency between 4500Hz and 4800Hz (half the sampling rate). Typically, humans can only hear between 20Hz and 20,000Hz. Since frequencies higher than 5000Hz are seldom seen in music, this sampling rate should avoid aliasing in most cases.

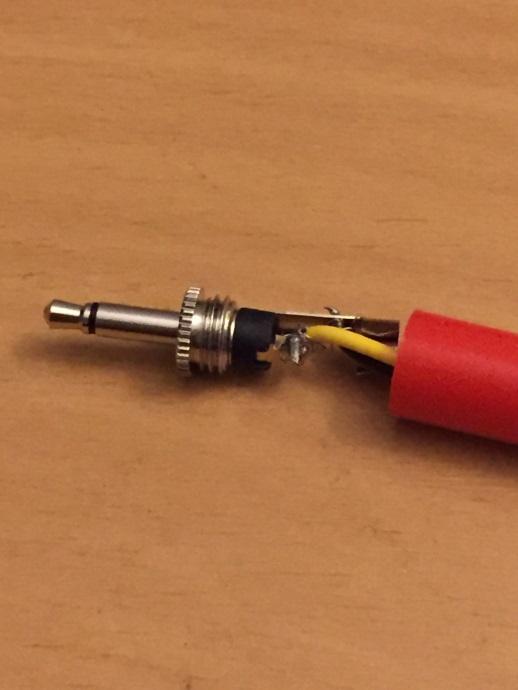
Real applications cannot sample infinitely, and must be limited to a window. For the FFT library used here, this window size should be about 7ms[2].  
  
**Technical Approach (Hardware)**

Figure 2 shows a block diagram of the system.   


**Figure 2- System Block Diagram**

As stated before, a laptop is used as the audio source. Most computers will mute speakers while headphones are plugged in. The ability to play audio through both speakers and headphones simultaneously is dependent on the laptop’s drivers and operating system─ because of this, a stereo splitter is used and an external speaker is attached. This way, audio can still be heard even when using the headphone jack for the spectrum analyzer.

Since stereo audio is not needed for the spectrum analyzer, a mono cable is used instead. Figure 3 shows the two soldered connections for the mono cable; the yellow wire carries the signal and the black wire is grounded.



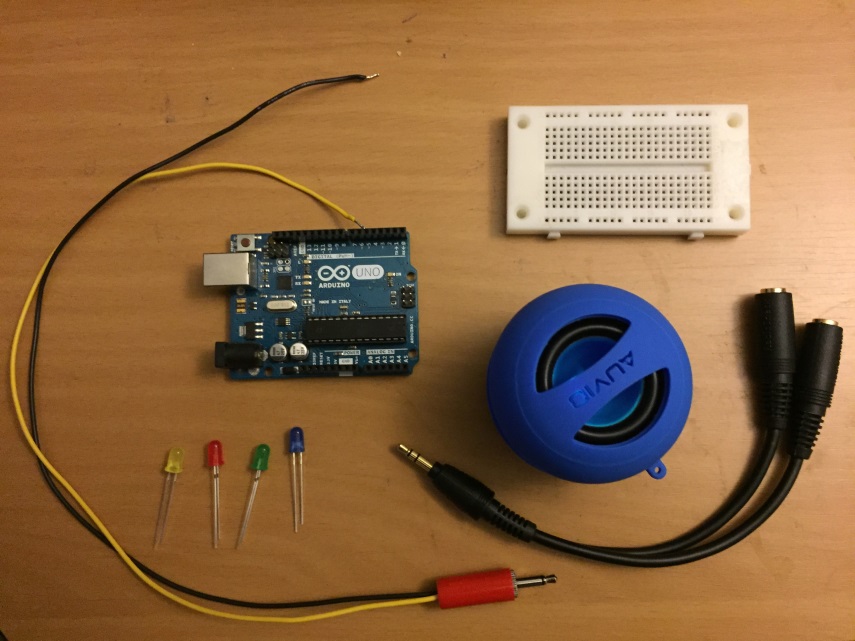
**Figure 3- Exposed Mono Cable**

The Arduino Uno cannot handle negative inputs, so the mono cable is inserted into a voltage divider to add a DC offset to the signal. This offset does not need to be subtracted since the DC component is not needed for analyzing music. The voltage of this signal is large enough to not require an amplification stage.

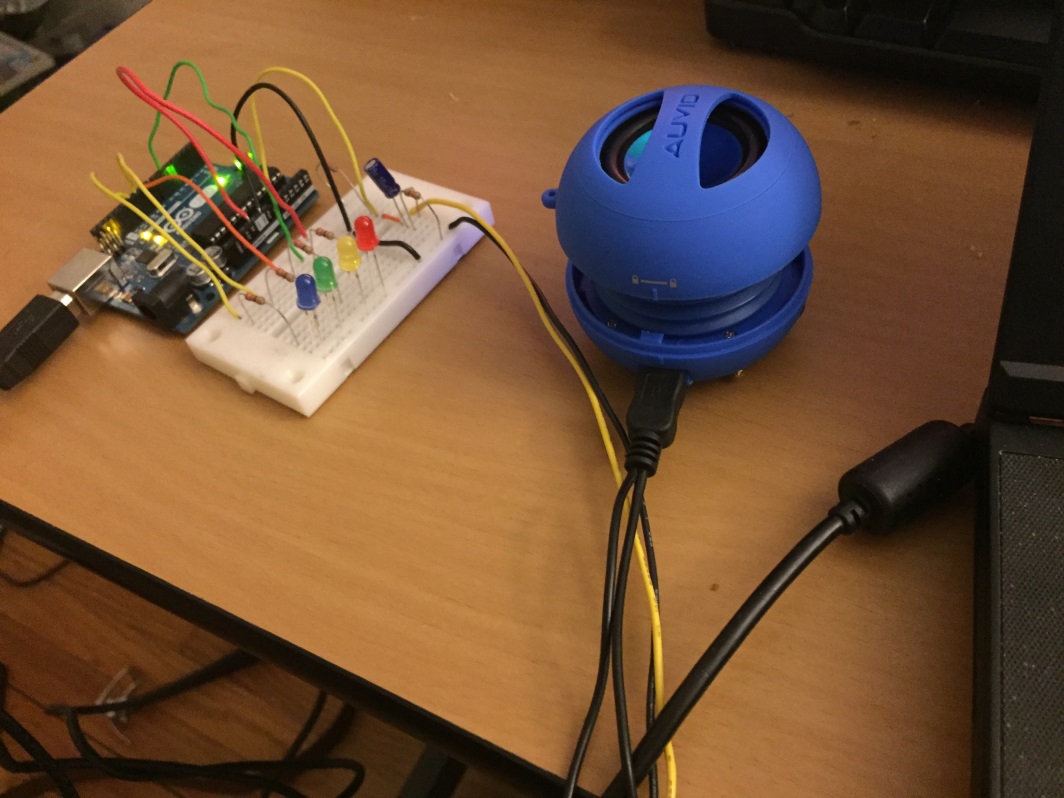
Many different microcontrollers can be used for this project, but an Arduino Uno was chosen because it is relatively easy to use, inexpensive, and widely documented. The Arduino can also receive power from the laptop, communicate serially with the laptop, and output enough current to light a small number of LEDs without an external power source. Unfortunately, the Arduino has some downsides as well; the ATmega328 on the Arduino has limited processing power, which makes it difficult to process a large amount of samples. Fortunately, there exist FFT libraries that are optimized for the Arduino, which makes this project possible. The Arduino also has some drawbacks to its inputs and outputs; the inputs cannot handle negative values, and the outputs can only handle digital outputs of 5V or 0V. These drawbacks have significantly altered the design of the spectrum analyzer and, if the project were to be scaled upward, the Arduino would likely need to be replaced.

After signal processing, the digital outputs on the Arduino are used to light up a Red LED, a Yellow LED, a Green LED, and a Blue LED. Since LEDs are semiconductor devices, their brightness are controlled through Pulse Width Modulation (PWM). The Arduino Uno can use PWM on six of its digital outputs, but this project only uses four LEDs for the sake of using unique colors. Each LED represents a range of frequencies, with blue being the lowest and red being the highest. When large magnitudes are present on the FFT output, the duty cycles of the corresponding LEDs increase and the proper LEDs light up. Current limiting resistors are used to protect the LEDs and Arduino.

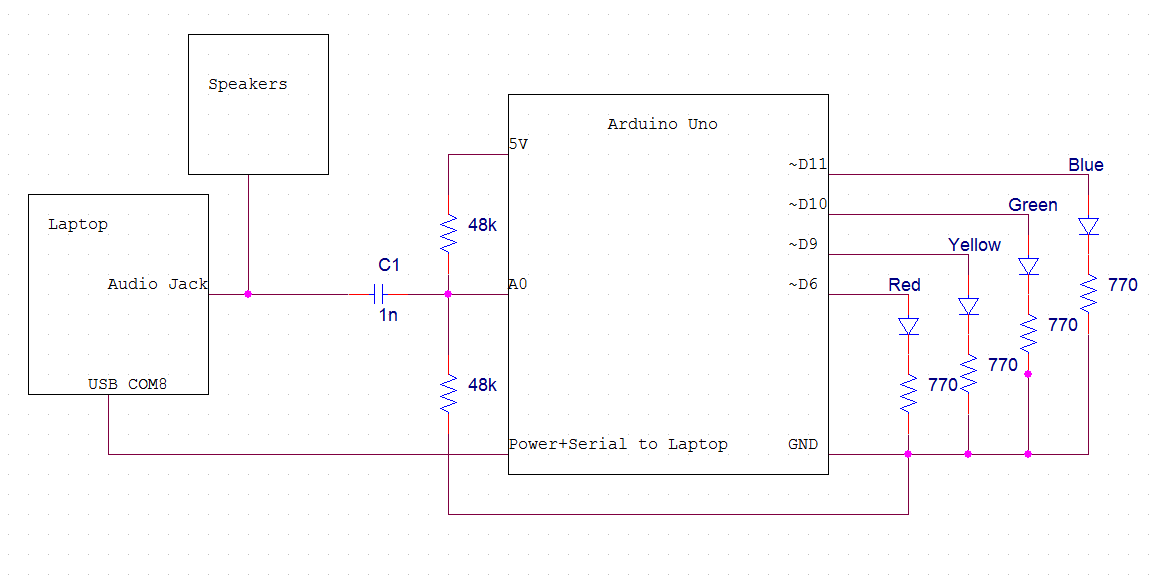
Figure 4 shows the different equipment being used for the spectrum analyzer, including the stereo splitter, speaker, mono cable, Arduino, LEDs, and a breadboard. Figure 5 shows the completed system with everything connected. Figure 6 shows a schematic of the system.



**Figure 4- System Components**

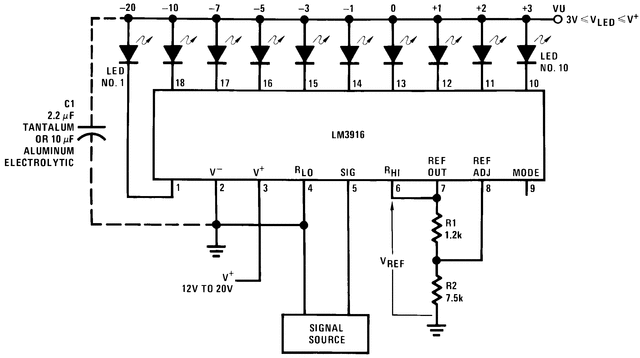


**Figure 5- System Configuration to Date**



**Figure 6- System Schematic**

A major hardware change was made since the project was first proposed. Initially, the magnitude of frequency spectra was to be represented using multiple LEDs driven by LM3916 ICs, which are Volume Unit driver chips. These ICs light up a number of LEDs corresponding to their analog inputs. Unfortunately, the Arduino is not capable of outputting analog values. Digital PWM can be used as a substitute for analog for many cases, but the LM3916 chips cannot function properly with PWM input. Simple circuits can convert PWM outputs to analog values, but the response times on such circuits are too slow for audio applications. Figure 7 shows the intended configuration of an LM3916 IC.



**Figure 7- LM3916 Configuration**

Fortunately, replacing these ICs with single LEDs does not change any of the software or DSP used in this project. The only difference is that, instead of representing spectrum magnitude with multiple LEDs, magnitude is represented by LED brightness instead.

**Technical Approach (Software)**

For signal processing, the ArduinoFFT library by Open Music Labs is used. Since audio needs to be sampled very quickly to avoid aliasing (given by Nyquist’s Criterion), picking a good library is necessary to get around Arduino Uno’s processing issues.

ArduinoFFT uses a modified version of Cooley Tukey FFT to find the frequency spectrum while using as few clock cycles as possible[2]. This modified FFT does a few key things differently than a standard FFT. First, since sine and cosine multiplications are expensive, ArduinoFFT checks to see when these values are 0 or 1, and adds instead. Second, lookup tables are used to calculate square roots. The overall order is still O(Nlog2N), but sees faster times for small values of N, which is perfect for this application.

For this specific project, a 256 point FFT is used, and since all data is real, only even bins are used, odd bins are zeroed, and the butterfly operations are processor friendly.   
  
 A useful feature that ArduinoFFT has is the option to create the frequency spectrum on an octave-based logarithmic scale. Sound and music are inherently logarithmic, so scaling the frequency spectrum by octaves is perfect for audio applications like this. For 256-point FFT, the bins are arranged as follows:

FFT\_N = 256 : bins = [0, 1, 2:4, 5:8, 9:16, 17:32, 33:64, 65:128]

Scaled for frequency, these bins have values closer to the following (given a maximum frequency of about 4600Hz):

FFT\_N = 256 : bins = [0:75, 76:150, 151:300, 301:600, 601:1200, 2401:4800, 4801:9600, 9601+]

Noting that the 0:75, 76:150, 4801:9600 and 9601+ are not common frequency ranges (a grand piano runs from 27.5Hz to 4186Hz), there are only four array values that are of interest. This feature provides a very easy way to decide the duty cycle for the LEDs; since there are four LEDs and four relevant array values, each LED receives its duty cycle from that value. The LEDs are assigned as follows:

Blue is about 151Hz to 300Hz

Green is about 301Hz to 600Hz

Yellow is about 601Hz to 1200Hz

Red is about 2401Hz to 4800Hz

These approximate values were found through using frequency sweeps as inputs.

ArduinoFFT can also communicate through serial output. With this, the Arduino can communicate with the laptop to provide spectrum magnitudes for human visualization. This project uses a baud rate of 115200Bd, but this can be configured to any values compatible with Arduino Uno, including 300Bd, 1200Bd, 2400Bd, 4800Bd, 9600Bd, 14400Bd, 19200Bd, 28800Bd, 38400Bd, and 57600Bd.

Outside of ArduinoFFT, the Arduino needs to do some other things in software for the spectrum analyzer to function properly─ that is, the Arduino must drive the LEDs with PWM according to their corresponding spectrum magnitudes. Because noise is present within the system, a 0% duty cycle is defined above average noise levels, and a 100% duty cycle is represented a spectrum magnitude of 256. The threshold at which a 0% duty cycle is defined is different for each LED. Appropriate thresholds were found to be 110 for blue, 85 for green, 75 for yellow, and 65 for red.  
  
**Intermediate Results**

Once the FFT code was written, data sampling was the next step taken in finishing the project. Data was sampled without any input to analyze the noise, and without any special scaling on the X or Y axes. The spectrum of noise without octave scaling is shown in Figure 8. Because of the periodic nature of the Discrete Fourier Transform, the noise spectrum looks very similar for all frequencies.

**Figure 8- Noise Spectrum**

The noise spectrum reveals a large DC component and an average of around 30. The DC component will not interfere with audio sampling, because values lower than 20Hz cannot be heard by the human ear. The average noise is useful in determining the level at which 0% duty cycle should lay.

Since music is logarithmic in nature, noise data was collected once more in octaves, using the OCTAVE feature in the FFT library. Figure 9 shows the octave noise spectrum. The octave spectrum is more useful for audio applications, so LEDs were driven using octave magnitudes. Note that the LEDs use the spectrum data on bins 2 to 4, 5 to 8, 9 to 16, and 17 to 32.

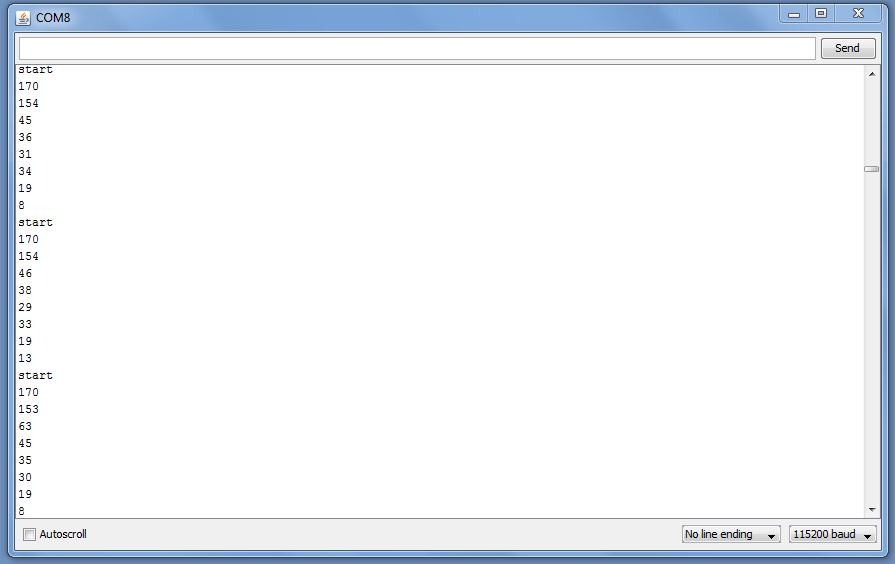
**Figure 9- Noise Spectrum (Octaves)**

Finally, to test the effectiveness of the algorithm, a sinusoid with a constant frequency of 2000Hz was used as an input. Figure 10 shows the resulting spectrum, which is the noise plus input frequency on an octave scale. This 2000Hz input increased the spectrum magnitude on octave 9 to 16, which corresponds to the yellow LED.

**Figure 10- 2000Hz Input (Octaves)**

The current algorithms properly capture data and provide expected results, but they do not do much to remove noise in the system. To compensate, LEDs are only lit once their bin magnitudes rise above a certain “threshold.” These thresholds were determined through rigorously testing different waveforms and song samples. Since songs tend to have more low-pitched sound than high-pitched sound, it was found that higher frequencies deserved smaller thresholds than lower frequencies.

Testing involved performing frequency sweeps and playing different songs. Some video examples are attached to the digital copy of this report. Also attached are spreadsheets containing the samples used for Figures 8, 9 and 10. These samples were taken through the serial monitor built into the Arduino IDE. Figure 11 shows an example of the serial monitor corresponding to the samples used to create Figure 9.

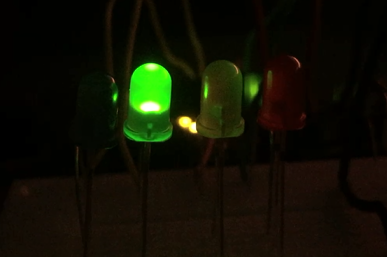


**Figure 11- Serial Monitor Samples**

**Final Results**

Overall, the spectrum analyzer works quite well. However, while testing, some oddities were noted.

The sound found in music and nature is often not as “ideal” as the sine waves used in testing. Many sound waves contain multiple frequency components that are not obvious to the human ear, but still show up in the frequency spectrum. For example, compare Figure 12 and Figure 13. Figure 12 shows the spectrum analyzer when fed a sinusoidal input with a frequency of about 800Hz. When the wave is purely sinusoidal, only the green LED is lit, as is expected. However, Figure 13 shows the result when a square wave of the same frequency is used. Since square waves have a sharp “drop-off” present, they possess higher frequency components that are not clear to the human ear. This causes higher LEDs to light up unexpectedly when inputs are not smooth.



**Figure 12- Sine Wave at about 800Hz**



**Figure 13- Square Wave at about 800 Hz**

For most cases, however, the audio spectrum analyzer works just fine when music is used as an input. Figure 14 shows an image showing the response to music. Note how each LED pictured has a different brightness corresponding to its octave magnitude.



**Figure 14- Spectrum Analyzer with Music**

The thresholds on each LED could still use more tuning to better match the visual with the audio, but any such changes would be minor and would not change the spirit of the project.

**Discussion and Further Work**

If given more time to work on this project, some major improvements would be:

* Adding a potentiometer to each LED to adjust the LED thresholds without changing code
* Selecting new equipment to build the analyzer with VUs, as originally intended
* Add more frequency channels
* Designing a filter to remove the noise present in the system

Most of these changes can be done just by adding more hardware, but a filter would be best completed through software. Since estimated noise is known (see Figures 7 and 8), a Wiener filter could be designed to minimize the mean square error between the noise and the desired outcome. If such a change were to be done, a new microcontroller would likely be needed for extra programming space and processing power.

**References**

[1] E.Zonca. (2015, March 23). *3D SPECTRUM ANALYZER USES 1280 LEDS* [Online]. Available:

www.http://hackaday.com/2015/03/23/3d-spectrum-analyzer-uses-1280-leds/

[2] Guest. (2014, July 08). *Arduino FFT Library*[Online]. Available:

http://wiki.openmusiclabs.com/wiki/ArduinoFFT