Music-LEDs:
Desrciption and Usage Guide

Matthew French

Updated: Novermber 21, 2018

As a Computer Engineer with a curiosity for circuitry, mircocontrollers, and software, I was inprired by both professional light shows accompanying concerts and a lack of how-to tutorials online to build a project with lights. I knew relatively little about any of the texhnology used when starting, but after the progress I've made, I can safely say I've learned quite a bit. Now, I want to share my project and experience with the world. I hope others enjoy it as much as I have, and I hope it insipires the same curiosity to go learn something too.

Though this project is my first major personal project, nevertheless it is one of which I am extremely proud. It is an arduino and circuitry project that can be wired parallel into an auxillary cord with a speaker to turn the pitch of music into a light show on a string of LEDs. Ill show you what I did and how it works so that you too can replicate and enjoy it.

Circuitry

The audio signal is wired parallel from an auxillary cable which is connected between an audio source and a speaker. The aux cable I used had wires for a sereo connection, but only one track (left or right) and ground is needed. The input to the circuit must be mono because of the noise when using stereo. The circuit itself is modeled after a "transistor pump frequency discriminator" from Thomas Hemingway's "Electronic Designer's Handbook" in Fig 13.5.[1] The elements for the ciruit were based upon LTSpice simmulation and available parts. The part numbers, resistances, and values for capacitors are labeled beside each component. Note, the capacitors are electrolytic. All of the components can be found online.

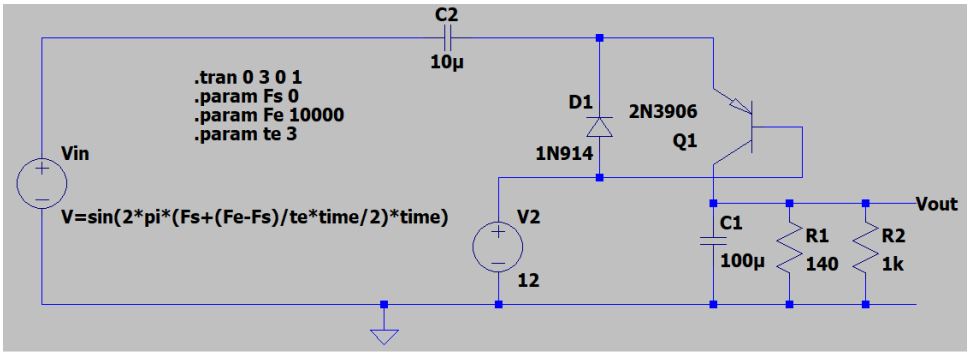

Figure 1: Schematic for pitch/frequency to voltage circuit.

The arbitrary behavior voltage source on the left hand side of the schematic represents the raw audio signal wired in parallel from the auxillary cable. The other voltage source is a constant

DC. Its value must be greater than the output voltage. The maximum voltage an arduino analog pin can receive is 5 V so a 5 V source is ideal, but I did not have one available. I used a cut 12V power supply wall adapter and cord that I had lying around for this purpose. Finally, I wired Vout and ground to an analog pin and ground (respectively) on my arduino.

To test this circuit's functionality, the voltage source which models the input audio signal is set to sweep through various frequencies. In the Figure 1 schematic, the parameters Fs and Fe can be changed to adjust the starting and ending frequencies which the source sweeps through. The variable te determines how long this sweep should take. I set this source to sweep from 0 Hz to 10 kHz in 3 seconds. The frequency of every piano key fits in this range.[2] I left the amplitude of this signal to be 1 V because the line voltages for every source I planned to use were lower than 1 V. For example, the the iPhone 7 and lightning adapter output a voltage of 440 mV for 200Ω loads and 430 mV for 16Ω loads.[3] Therefore, even as the audio source voltage approaches 1 V, the output voltage should stay below the maximum 5 V that can be read in by the arduino's analog pins. The figure below shows the output voltage as a function of time, and thus a function of frequency.

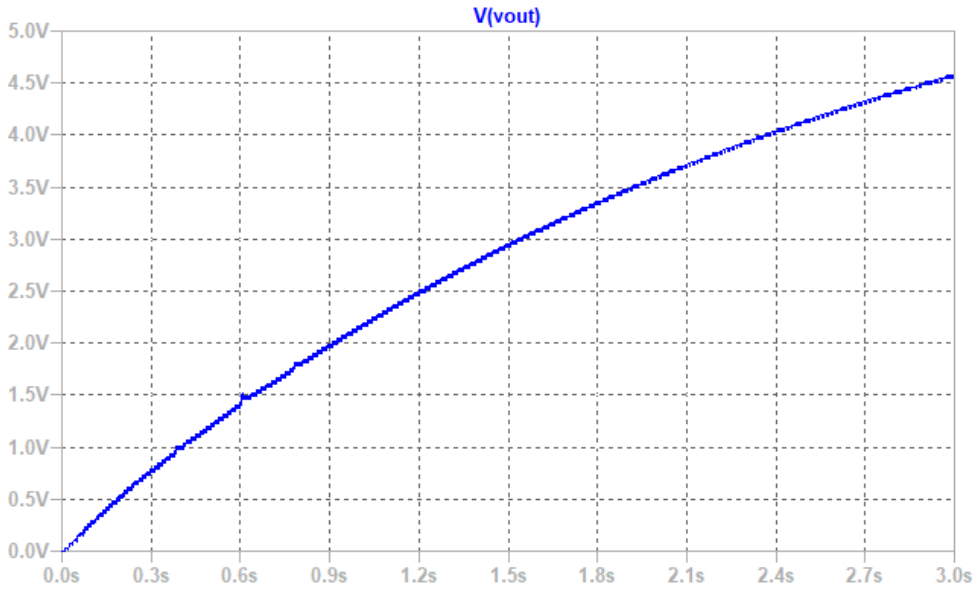

Figure 2: Simulation results of output voltage as a function of the input signal's frequency.

The figure shows the voltage level as the frequency changes. To reiterate, the voltage source sweeps through frequency with time. The frequency at 0 s is 0 Hz. The frequency at 3 s is 10 kHz. I made this sweep happen quickly to try to replicate how the pitch in a song and thereby frequency in an audio signal can change rapidly. Feel free to build the schematic to test other values of components to assure the functionality and safety of the circuit and arduino.

Though this plot suggests that the output voltage for low frequency signals should also be low, in reality the circuit does not behave exaclty this way. For example, in testing songs with hard-hitting bass, I found that often the output voltage would be higher than the simulation would suggest. I suspect a few culprits. For one, the simulation is continuous in its sweep of frequency. This is reflected the smooth output voltage curve. Because of the potentially discontinuous shifts in frequency in audio signals, the circuitry might behave differently. Another cause, which I suspect plays an even more significant a role is the variance in voltage of the audio signal. The circuit proposed by Thomas Hemingway assumes the input signal is a square wave of varying

frequency but constant amplitutde. Audio signals do not have a constant peak voltage to reflect a variety of volume levels. One reason the lower frequencies might cause a high output voltage is because they can be loud. The bass causes the audio signal to have a large peak voltage and therefore a larger than expected output voltage.

Though the reality deviates from simulation, nevertheless I have accepted the presence of this discrepancy and even welcomed its usefulness. For example, a high voltage at bass peaks makes for a more dramatic light show because the lights seem to react to both the pitch of the music and the beat. One negative result is that with a song with an overpowering, dominant bass, the primary reactivity of the lights with be due to the bass rhythm and the melody pitch will play a negligible role.

One more flaw of this circuit is its nonlinearity between frequency and output voltage. The simulation of the circuit using a smaller C1 and larger C2 led to a more linear relationship, but the availability of certain capacitor values limited my linearity. This is only a minor issue because the relationship between perceived pitch and frequency is not exaclty linear either.

Though the circuitry does have its flaws, nevertheless it was much cheaper than buying a frequency-to-voltage chip to do the entire process automatically. Plus, I learned a decent bit about how to design and wire circuitry, and how to build a simulation using LTSpice.

The other circuitry I used in this project was an array of TO-220 MOSFETs to control the RGB switching of the LED strip. That circuit is shown below and can be found on Adafruit's website.[4]

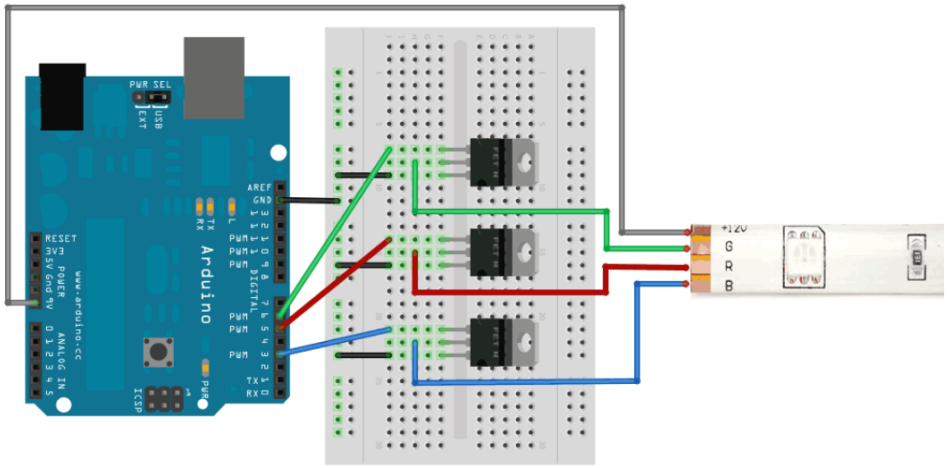

Figure 3: Schematic of the wiring to control the LED strip. Note, some changes were made to this schematic such as power distribution.

I did make a one change to the schematic given by Adafruit. For the given schematic, Adafruit suggests to wire the power for the LED strip via the 9 V (Vcc) pin on the arduino. I decided to separate all power from the arduino for a few reasons. For one, the LED strip I bought required more power than I wanted to supply via the arduino. By wiring the power for the LED strip externally, I could choose the power supply that was necessary to power the lights without having to supply my arduino with the same power. In addition, by making the supply voltage for the frequency-to-voltage circuit 12 V, I could use it also to power the LEDs. Secondly, I decided, in general, to keep all power off-board if possible. Because I am still new and learning how the circuitry works, this ensured that I did no damage to the board via power supply. To

connect power, I simply made a connection from the +12V terminal of the LED strip to the +12V power supply lead and a connection from the arduino's ground pin to ground of the power supply and input audio signal.

Algorithm

I found a suprising difficulty when working on the argorithm to control the LED strip. This difficulty arose due to the nature of the RGB color scale. The figure below shows the breakdown of R, G, and B values which make up every color on the RGB scale.

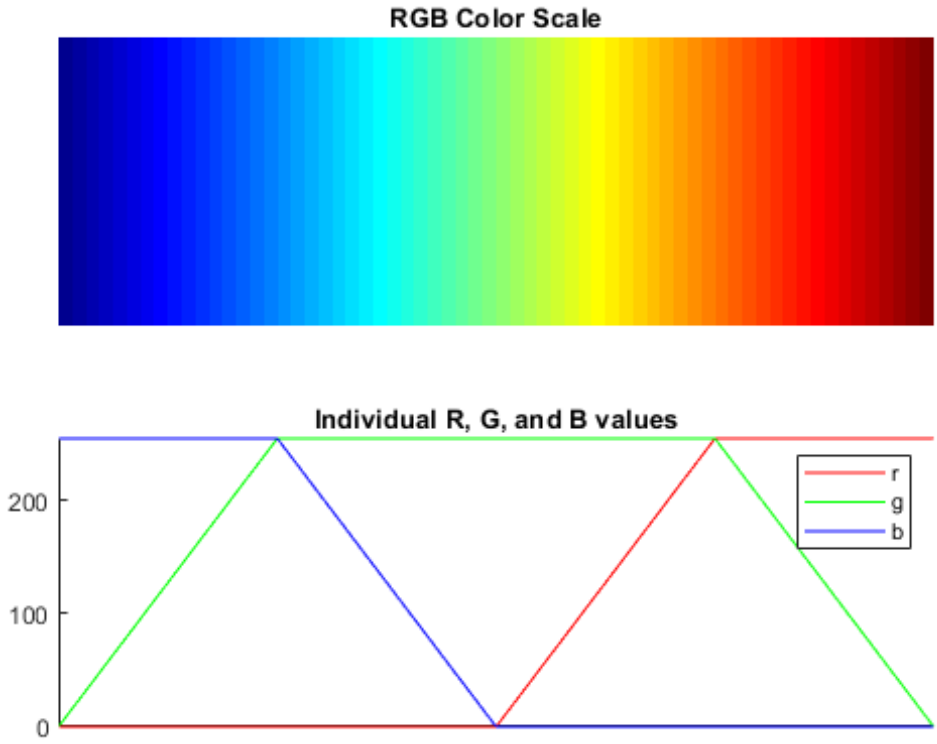

Figure 4: The top graphic shows the scale as a whole. The bottom graphic shows how each color in the scale above would be represented in RGB.

The difficulty came because the method for traversing the scale varied depending on the present location of the scale. For example, to change the LEDs from green to red, r must increase while g must decrease. However, to change the LEDs from blue to red, r must increase while b must decrease. Also, the behavior differed depending on the destination color. For example, to change the LEDs from blue to red, r must increase while b must decrease. However, to change the LEDs from blue to green, g must increase while b must decrease. I experimented with a number of different algorithms to some success, but no algorith worked better than breaking the behavior into cases.

To begin, each song and audio signal is different. For example, some songs have mostly low pitch with many bass notes. Others have higher pitched melodies. Because of this, defining a scale needed to be dynamic. Each loop reads in a value from the frequency-to-voltage circuit. To spread apart values in the higher frequency range where the nonlinearity of the circuitry makes the differences small, an exponential power greater than one is applied. That value is compared to and updates a max value. From this max value, the color spectrum is broken into five intervals. This is because the scale I used started at red, progressed through green and blue,

and ended at blue-red, in essence wrapping back to the beginning. The number five was chosen becauuse for the individual RGB value behavior, the points at which the piecewise behavior of one interval met another defined a change in behavior. For example, between the transition between red and green, r increases and g is constant, then r is constant while g increases. There are essentially two different behavior between purely red and purely green, so two intervals were chosen for this portion. I found that the size of each interval needed to differ slightly from a constand one-fifth of the max value due to the nonlinearity of the circuit, and the frequency of different signal values.

There is also a previous signal value which is saved at the end of each loop. The current signal value is compared to this previous value to determine the proper behavior for the LEDs. Because there are five intervals for the previous value, and because there are five intervals for the current value, the resulting number of cases was twenty-five. The rest of the algorithm is simple. The type of transition is determined based on the previous and current value and the r, g, and b values are adjusted accordingly. Finally, the r, g, and b, values are sent as an output via the PWM digital pins to the MOSFET array which controls the LED strip.

References

1. https://archive.org/details/ElectronicDesignersHandbook
2. https://en.wikipedia.org/wiki/Piano_key_frequencies
3. https://ifixit.org/blog/8448/apple-audio-adapter-teardown
4. https://learn.adafruit.com/assets/2692

- http://mathscinotes.com/2014/03/a-simple-frequency-to-voltage-converter
- https://electronics.stackexchange.com/questions/291721/how-to-implement-frequency-sweep-in-transient-mode-in-ltspice
- https://www.youtube.com/watch?v=lU1GVVU9gLU&t=4s