Frequency-responsive RGB Instrument Lights

Anja Frohock, Carson Warner, Leith Rabah, Stefan Gonzalez

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — Frequency-responsive RGB (FRGB) Instrument Lights are a type of lighting system that allows users to synchronize the lights with the music or sounds in the room. This technology is based on the concept of sound-to-light conversion, where the frequency and volume of the sound are translated into different colors and intensity of light. The FRGB lights typically consist of red, green, and blue LEDs that can be mixed to create a wide range of colors. The conversion is done through an application that requires a microphone to pick up the sound and translates it into signals that control the FRGB lights. This technology can commonly be used in music festivals, concerts, clubs, and home entertainment systems. It adds a unique and dynamic visual element to the music or sound experience, enhancing the overall sensory experience for the audience.**

***Index Terms* — Analog to digital conversion, color, fourier transforms, frequency conversion, instrument, music, signal to noise ratio.**

I. Introduction

FRGB Lights are an interactive system that turns on LEDs at the notion of the playing of an instrument, each instrument will generate its own distinct response based on the notes being played. Each instrument has its own distinct logistical challenges in which the input and output will vary. The complexity of the project increased with the involvement of more instruments, the portability of the interactive system, and the intensity of the LEDs responses based on the volume and pitch. The system displays light intensities based on the individual instrument's wave response when played at different frequencies. FRGB Lights are relatively low cost, portable, require low power, and are easy to use. As there are currently no well established products of a similar nature, there is an open market for it to take. The use of Raspberry Pi, specifically Pi4, is the brain of the product. FRGB instrument lights have the potential to raise the quality of musical performances and the experience of playing the instruments. The responsiveness of the lights to the pitch and volume of the music acts as a visual aid for both users and observers. Outside of enhancing entertainment facilities, this visual aid can be beneficial to anyone ranging from those that lack the skills to interpret perfect pitch to those that are hard of hearing or even deaf. It can also be used as a method of displaying the daily occurrences of people with a form of synesthesia where they can see music in the form of colorful lights.

II. Software application

FRGB instrument lights are controlled by a Raspberry Pi 4 microcontroller which runs a python application. This application utilizes the pyaudio, neopixel, and tkinter libraries, the pyaudio library enables python to read information from an externally connected microphone such as pitch and volume, the neopixel library is a library that is used to control addressable RGB LED strips in color and brightness of each individual LED. These libraries are used in conjunction with each other in order to convert the sound frequencies being read by the microphone into numerical values for frequency and volume. These values are then used to determine the musical notes and the color each note corresponds to which is then displayed using the LEDs. This application comes with a functioning graphic user interface (GUI) which is enabled by the tkinter library. The tkinter library introduces the ability to create a GUI in python with objects such as labels, buttons, frames, and much more. The GUI lets the user interact with the device through a small LED screen attached to the microcontroller. The GUI can be used to customize the color of each individual musical note and the intervals between them and also give visual feedback to the user of the current note on the chromatic scale that is being played.

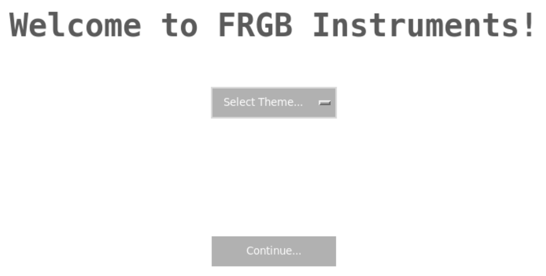

Fig. 1. Welcome screen of Python GUI

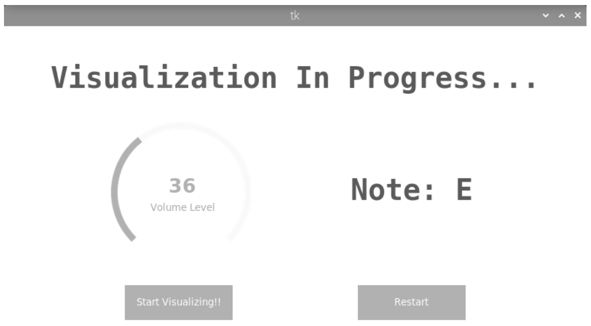

Fig. 2. Program in progress screen with real time note information

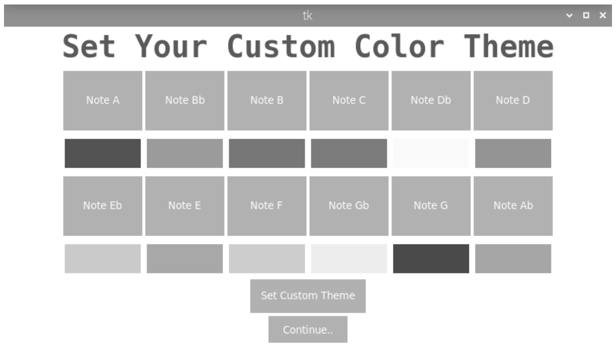

Fig. 3. Custom theme selection screen

III. Fourier transforms

Fourier Transform (FT) is a mathematical technique that transforms a function of time to a function of frequency [1].

$$F\{g(t)\} = G(f) = \int g(t)e^{-i2\pi ft}dt \quad (1)$$

In the case of FRGB lights, FT is used to convert the audio signal from the time domain into the frequency domain. so that the frequency spectrum can be analyzed and used to control the RGB colors of the lights. This conversion is necessary because the sound signals are represented in the time domain as a series of amplitude variations over time, while the frequency domain represents the sound signal as the sum of individual sine waves of different frequencies and amplitudes. This calculation involves breaking down each segment of the sound signal into its component sine waves of different frequencies, amplitudes, and phases.

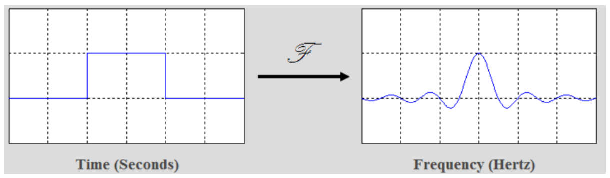

Fig. 4. Fourier Transform Visualization

This is done through the Fast Fourier Transform algorithm. Fast Fourier Transforms (FTT) is a programming algorithm that computes the FT in an O(N(log2(N))) runtime. This allows us to conduct Fourier transforms and dynamically change the colors of the lights in what can be considered real time. The result of the FFT calculation is a set of numbers that represent the amplitude of each frequency component in the sound signal. These numbers can be used to analyze the frequency content of the sound being emitted into the microphone and map the frequency spectrum to the corresponding RGB colors.

The audio signal is first captured by a microphone and sampled at a fixed rate of 0.1 samples per second, producing a series of discrete data points representing the amplitude of the audio signal at different points in time. The output of each sample is a set of complex numbers that represent the magnitude and phase of each frequency component in the audio signal. The magnitude represents the amplitude of the frequency component and the phase represents the relative position of the component in the time domain. This information is utilized in the FRGB lights in two ways. The first is by using the phase to determine and adjust the colors of the lights and the second is by using the magnitude as volume to determine the intensity of the lights.

Fourier transforms are the backbone of the FRGB lights. Without this algorithm, the software program would not be able to run and conduct the necessary calculations in order to control the lights.

IV. Hardware implementation

FRGB instrument lights utilize many different pieces of hardware to achieve its goal of music visualization. The main essential hardware components include a Microcontroller, Microphone, DC-DC converter, LED strip, ADC, Microphone Pre-amp, and an LCD touch screen display.

A. *Microcontroller*

For our microcontroller we chose the Raspberry Pi 4 due to its relatively high speed for its size, cost, and its overwhelming community support. The Raspberry Pi 4 was in contention with the Nvidia Jetson Nano when choosing the microcontroller due to its higher CPU speed however when weighing the options the Nvidia Jetson Nano while faster than the Raspberry Pi 4 is also twice as expensive so to reduce costs and keep the project simple we chose the Raspberry Pi 4.

B. *Microphone*

The microphone was required to pick up as little background noise as possible. This required a unidirectional (cardioid) microphone in order to pick up unwanted noise from any direction other than where the instrument was being played.

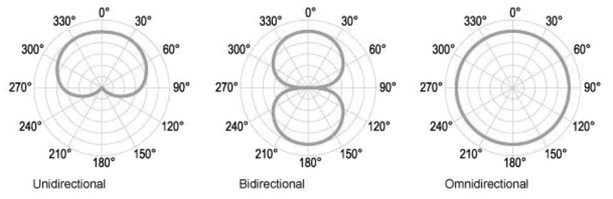

Fig. 5. Unidirectional vs. Bidirectional vs. Omnidirectional

Initial testing also required the microphone to be able to directly connect to the desired instrument. With these in mind, a clip-on condenser microphone was chosen for initial testing, specifically the YPA M613-XLR.

C. *Phantom Power supply*

To accompany the microphone we chose a pre-amp to amplify the microphones output to a reasonable level to analyze, with condenser microphones we needed a certain type of preamp with phantom power support which sends power to the microphone since condenser microphones require power to operate. We chose the Xvive P1 Phantom Power Supply which fulfilled our needs as a phantom power preamp with the added bonus of being battery powered to reduce cable clutter.

D. *LED Strip*

The most important feature when selecting an LED strip to use for the product was RGB capabilities. For this reason, we chose to use SMD light strips for the initial prototype of the project. Other important aspects such as brightness, density, and programming support were considered when selecting a light strip. With these attributes in mind, and inspiration taken from RGB computer keyboards, we chose the BTF-LIGHTING WS2812B which is also supported in the neopixel library which was a large factor in LED choice since the library only supports a certain type of LED strip. These strips also had the lights placed densely together, giving off a more uniform appearance as if it was one long light. Finally, the brightness of the lights were of a high enough lumens to be visible from anywhere in the room even with the lights on.

E. *LCD Touch Screen*

The LCD touchscreen only had three requirements, those being that the screen is touch sensitive, that it is compatible with the Raspberry Pi 4, as well as being within the size requirement of 7x5x5 inches for the design of the box. With these parameters in place we chose the waveshare 5 inch DSI LCD Touch Screen which met all of the aforementioned requirements. This screen is used to assist the client by allowing them to interact with the software application and adjust the settings and themes of the lights as they see fit.

F. *Pre-Amplifier PCB*

The Preamplifier circuit was designed to take the output of the mic after the phantom power supply and amplify it from mic level to a voltage that would work with the analog-to-digital converter. Originally the mic level was measured to peak at 50mV. However, the input used wasn't loud enough so when it was measured again it peaked at 200mV. Originally as well the gain was designed to be 50 to multiply the original peak of 50mV to 5Vpp. The Raspberry Pi GPIO pins only take a maximum voltage of 3.3V. That with the remeasured peak of 200mV led us to design a pre-amplifier with a gain of 15.

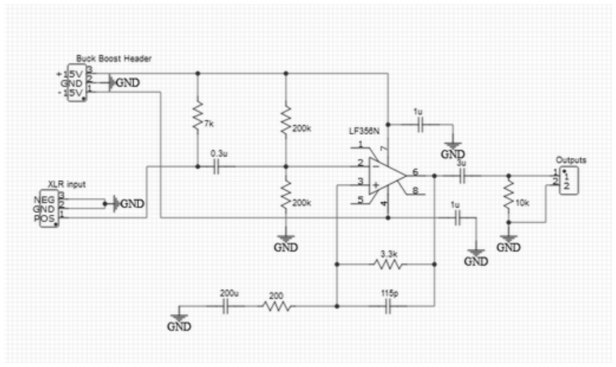

Fig. 6. Pre-Amplifier Circuit

G. *Analog-to-Digital Converter*

Analog to digital conversions can be done in various forms with their general design goal of converting a difficult to store analog signal to an easier to process and store binary digital signal. The design chosen was the delta-sigma architecture rather than the dual-slope, flash, pipelined, and the successive approximation architecture. Reason being this is the best architecture to handle low frequency applications, however, there is a downside of higher latency due to oversampling. For this project the ADS1115 was used.

H. *Device Casing*

The device had to maintain a guide-line of 7x5x5 inches in dimensions and the weight being under, if not equal to, 3 pounds. In which caused the rise of various options in making a case for the electronic components of our device. The casing had to be small, durable, and adaptable to changes. Reason for small is due to grand desire to have the device directly mounted onto the user, and be easily handled. Meaning that the User should be able work with the device within their hands. The goal for durability is to have the device withstand the vibrations, heat, and kinetic forces that come with the mounting of the device onto the User. The Device's sensitive electronics like the LCD and the microcontroller are safely placed and secured with screws. The last is having the case be adaptable with the possibility of additional components or to adjust the placements of current components. The reason for rearrangement is to help improve thermal levels within the device and in order to prevent the components from colliding with each other and to ensure no misconnections between the PCBs. The Casing will consist of standard PLA material and shaped to be a box shape. The final design changed from the original design to include small air vents to allow air flow within the device, reducing the height of the case from 5 inches to 3 inches and also having the stand to be its own piece to insert into the case's frame. This is to reduce the cost of printing and be able to modify the case better if issues occur

V. Sigma-Delta Analog-to-digital Converter Design

The Sigma-Delta analog-to-digital converter was designed to take the microphone`s output and convert it into a digital signal to be sent to the Raspberry PI via SPI. To make the project more technically advanced it was decided to design and build one. The sigma delta was chosen due to its low quantization noise and noise shaping properties. The noise shaping helps as it pushes the noise to higher frequencies. This is shown in figure 9. This is ideal for the project as only the first 20kHz is needed as that's what the microphone will pick up.

A. *The first Design*

The designed ADC has 4 different stages, The first is the difference amplifier and integrator which is repeated to create the second order ADC. The difference amplifier subtracts the previous signal from the input. originally. The integrator is what does the noise shaping for the ADC. originally THS4222DGK op amps were used for both. The next stage has the comparator, which is a 1-bit ADC to quantize the signal which will be our output. A MAX907 EPA+ was used for this. The next stage is the DAC, this is where the signal is fed back into the input. To begin the ADC has a D flip-flop the SN74LVC1G80DBVR that injects the clock into the signal. Next used is a switch, the MAX4544 CPA+ to act as a 1-bit DAC. Then an inverting op amp to flip the signal. The fourth and final part is the Decimator/digital filter that decimates the oversampling and filters out any noise. An ACPL-0873-500E is used for this. All parts except the decimator/digital filter are shown in the schematic shown in figure 7.

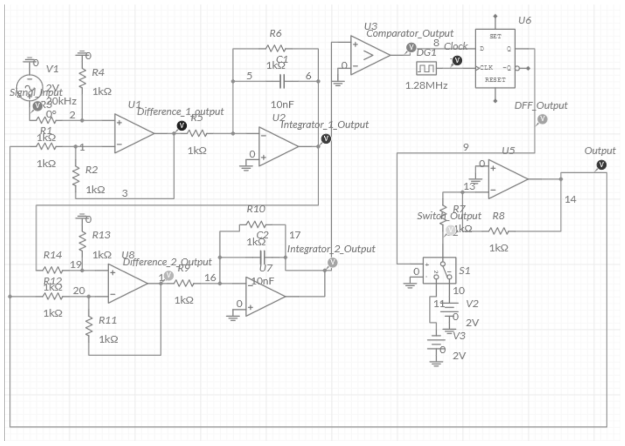

Fig. 7. Sigma-Delta Modulator Schematic

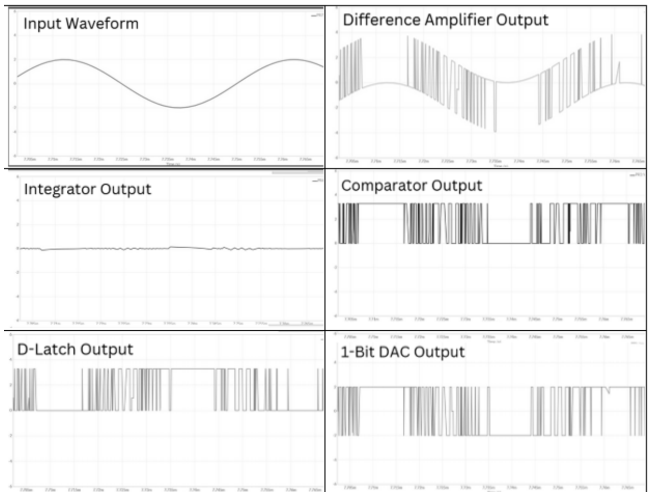

Fig. 8. Sigma-Delta ADC Simulations

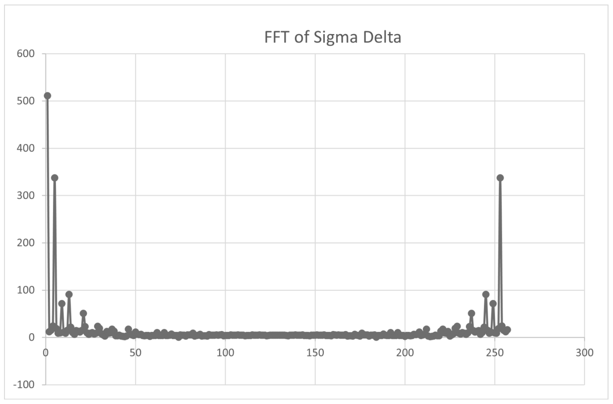

Fig. 9. Frequency Spectrum of Sigma Delta ADC

While trying to build this circuit many problems occurred. This was most likely due to using multisim live which only had ideal components rather than the components actually being used. When testing there were problems with the op-amps selected. This was most likely due to trying to use the same op amps for all of the sections instead of choosing op amps that suited each function. However, many op amps were tested and none of them worked. Another theory was that decoupling capacitors were forgotten but even after it didn't give any output. Overall there were too many variables to test in the timeline we were given.

B. *The Second Design*

As there is a deadline for this project it was suggested to try a design that had already proven to work. For that an MIT project was chosen as they built a sigma delta ADC PCB for their project [2].Their design was similar to the one designed for our project but they included a sample and hold circuit as shown in figure 10. Their output for the simulation is very similar to what the original design was outputting for the simulation as shown in figure 8. They also included voltage buffers. The parts used for this were an LM741 for the difference amplifier, LF356s for the integrator and subsequent op amps, Lm311 for the comparator, and BUF634AIDR for the voltage buffers. Also 2N7002 MosFETs. They used a 60MHz digital clock however this project is using an 8MHz oscillator adapted to a square wave and then pushed through an 8 times multiplier to create a 64MHz clock.

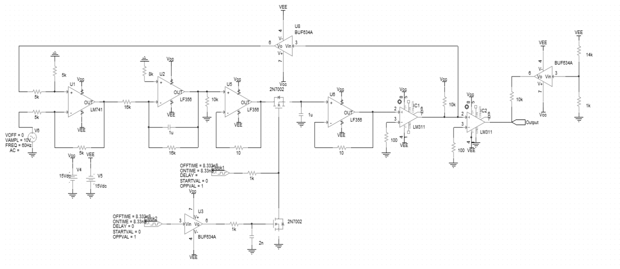

Fig. 10. Second Design Sigma-Delta ADC Schematic

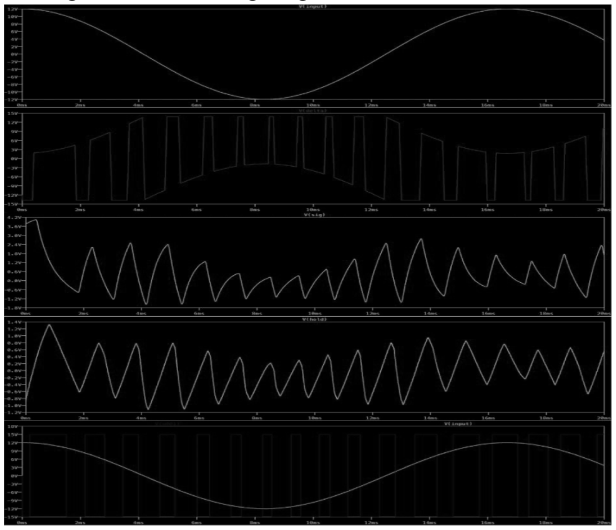

Fig. 11. Second Design Sigma-Delta ADC Simulation

At testing there were many issues with the comparators only outputting 1V. This was most likely due to the voltage buffers not working properly. The design used the OPA633 as voltage buffers however, they have since been discontinued. Therefore, they were replaced with the BUF634AIDR which was recommended by digikey and other sources. When testing the buffers did not output anything. This would explain why the comparator was only outputting 1V as it wasn't getting any input from the sample and hold circuit. This was especially hard to test as the buffer required a minimum bandwidth of 35MHz which was too high for the function generator to output. This would also explain why the rest of the circuit was incorrect as the other voltage buffers would not receive the minimum bandwidth and therefore not work as well as the feedback loop inputting the wrong values back into the circuit.

C. *ADS115*

As the deadline was only 2 weeks away Dr Chan was consulted on possible issues, he recommended we use an ADC IC instead. For that the specifications were that it needed to have a high resolution of at least 16-bits and have a high SNR value as both of those are very important when converting audio signals. The ADS1115BQDGSRQ1 was chosen as it had libraries for the Raspberry PI as well as a module by ADAFRUIT. The ADS1115 has I2C pins to communicate the output directly to the Raspberry PI. The module was used to assure both the breadboard and PCB were working as desired. As shown in figure 12 the ADS1115 PCB did work for the final project and was used to display the volume of the music being played on the LCD.

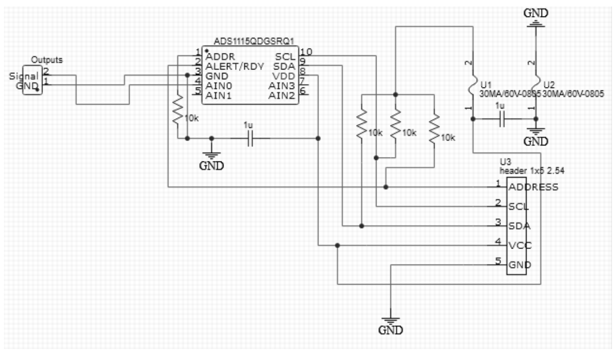

Fig. 11. ADS1115 Schematic

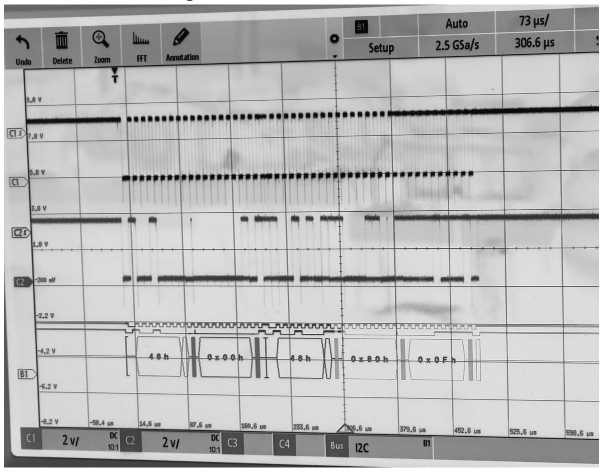

Fig 12. ADS1115 Output

VI. System Concept

This software diagram acts as a visual aid to help show the relationship of all of the components of the program that handle the audio to visual conversion. The block diagram shows the relationship between the software parts of the project and which blocks of the program are directly dependent on each other and how they work if one condition is met or is not met with the subsequent actions of the program based on the conditions.

The subsequent hardware diagram acts as a visual aid to help show the relationship of all of the components of the device that handle the audio to visual conversion. The block diagram shows the relationship between the hardware components of the project and which components of the device rely on other components to gain functionality.

A. *Software component relation breakdown*

The components of the software that run the device have many different interactions and stages that must be met to successfully turn the audio input into visual output. To start the program introduces a GUI which lets the user choose a color scheme for audio visualization, if the color scheme is selected then the driver code will be initiated if not the GUI will wait for user input. Once the driver code is active the program establishes a frequency and volume lookup table for later reference after the audio signal is processed. After this the unprocessed input audio signal is passed through a fourier transform and then the values given are compared with the lookup tables previously established. If the values do not match those in the lookup table then the program will revert to converting the audio signal again and if the values do match then the values from the table are directly output to the LEDs that are connected through a built-in python library.

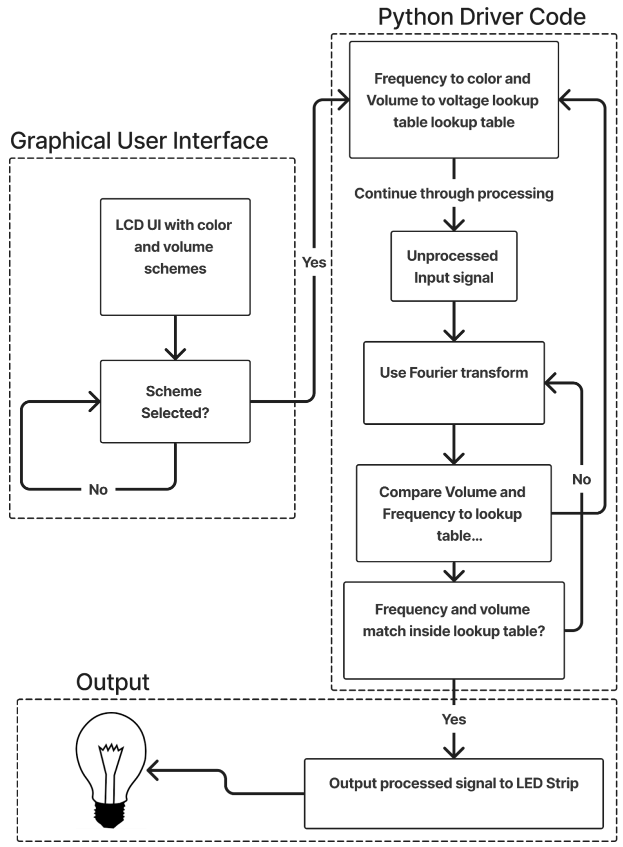
Python Driver Code
Graphical User Interface
LCD UI with color and volume schemes
Scheme Selected?
No
Yes
Frequency to color and Volume to voltage lookup table lookup table
Continue through processing
Unprocessed Input signal
Use Fourier transform
Compare Volume and Frequency to lookup table...
No
Frequency and volume match inside lookup table?
Output
Yes
Output processed signal to LED Strip

Fig. 13. Complete software system block diagram that shows logic behind audio to visual conversion.

B. *Hardware component relation breakdown*

The hardware components that make up the device interact with each other in many ways in order to produce the desired output effect onto the LED strip. To start we have the components that are directly attached to the instrument which are the LED strip and Microphone, the microphone is powered by an external Preamp and the microphones output signal is sent through a hardware ADC converter. The Pre amp ADC and all subsequent components are housed in a container to reduce clutter and increase portability. The output signal from the ADC converter is connected directly to the Microcontroller via GPIO pins. The microcontroller is subsequently connected to an LCD screen via DSI cable and the LED strip via GPIO pins. The LCD screen, microcontroller, and LED strip are all powered by a DC-DC converter which draws power via a US standard 2 Prong power supply plug.

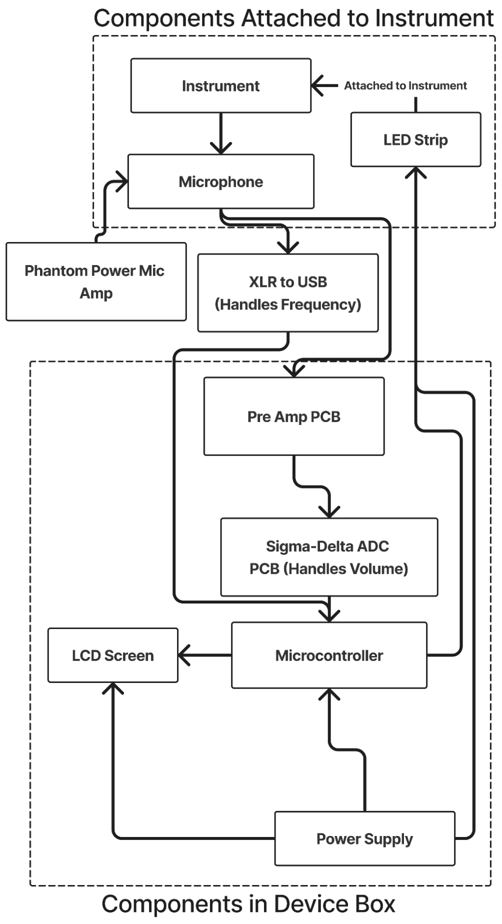

Fig. 14. Complete hardware system block diagram that shows the components behind audio to visual conversion and their relations.

VII. Conclusion

In conclusion, the FRGB lights provide an exciting new way to visualize music and sound through a combination of RGB lighting and sound analysis techniques. By using a microphone and a simple software application, the frequency and volume of the sound can be analyzed and used to control the lights in real-time, thus creating a dynamic and responsive lighting experience that is synchronized with the audio.

The FRGB lights offer a unique opportunity to explore the relationships between sound and sight, and to create stunning visual effects that are tailored to each user's individual artistic preferences. The lights can also be used to visualize some of the effects of synesthesia, providing a new and immersive way to connect and better understand the neurological condition.

Overall, the FRGB lights represent a significant development in the field of music visualization, providing an accessible and affordable platform for artists, musicians, and enthusiasts to create unique and personalized lighting effects that are synchronized with their own sounds.

Acknowledgement

The FRGB lights team would like to thank Dr. Richard Leinecker for sponsoring our project and supporting us on this journey.

References

[1]
https://lpsa.swarthmore.edu/Fourier/Xforms/FXformIntro.html
[2]
https://web.mit.edu/6.101/www/s2020/projects/colinpc_Project_Final_Report.pdf