FRGB Instrument Lights

Group 25

Sponsored By:
Dr. Rick Leinecker

The Team

Leith Rabah
Computer Engineer

Anja Frohawk
Electrical Engineer

Carson Warner
Computer Engineer

Stefan Gonzalez
Electrical Engineer

Introduction

- Frequency-responsive RGB (FRGB) instrument lights are RGB lights that are designed for musical instruments. The LEDs are controlled by the frequency and volume of the instrument being played where the frequency determines the musical note which determines the color of the LED and the volume determines the brightness of the LED.

- The main goal was to create an immersive visual element to musical performances in order to better captivate a wider audience.

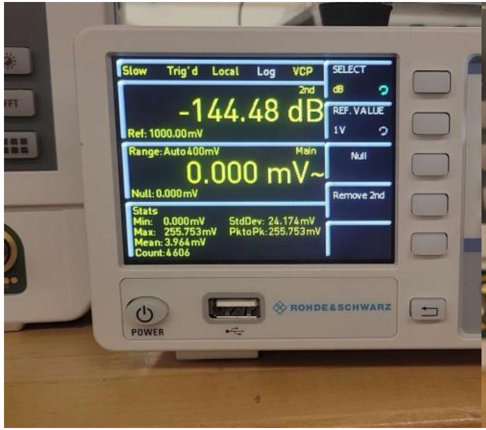
Specifications

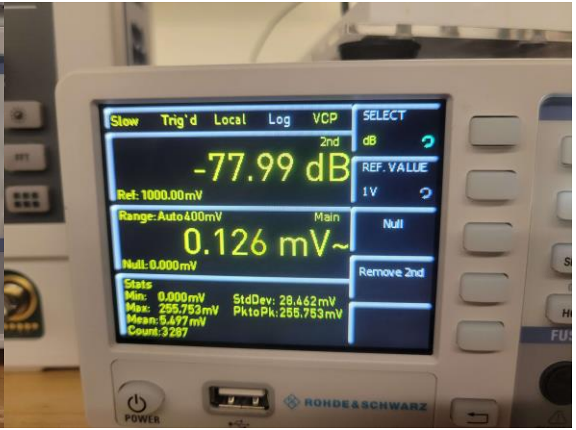
Specification	Goal
Minimum SNR value	25 dB
Minimum refresh speed	5 iterations per second
Fully customizable note colors	Can customize the color associated with each note individually
Portability	≤ 7x5x5 inches ≤ 3 lbs

Signal-to-Noise Ratio

- When calculating the SNR, the labs did not have the required equipment to accurately measure the values for the calculations.
  - Lab head David Douglas assisted with finding the values using the multimeter despite it not being as accurate as with a Decibal Reader
- SNR values:
  - Signal: -78 dB
  - Noise: -144.5 dB
- SNR = -78 - (-144.5)

  = 66.5 dB

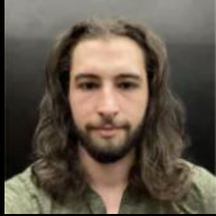




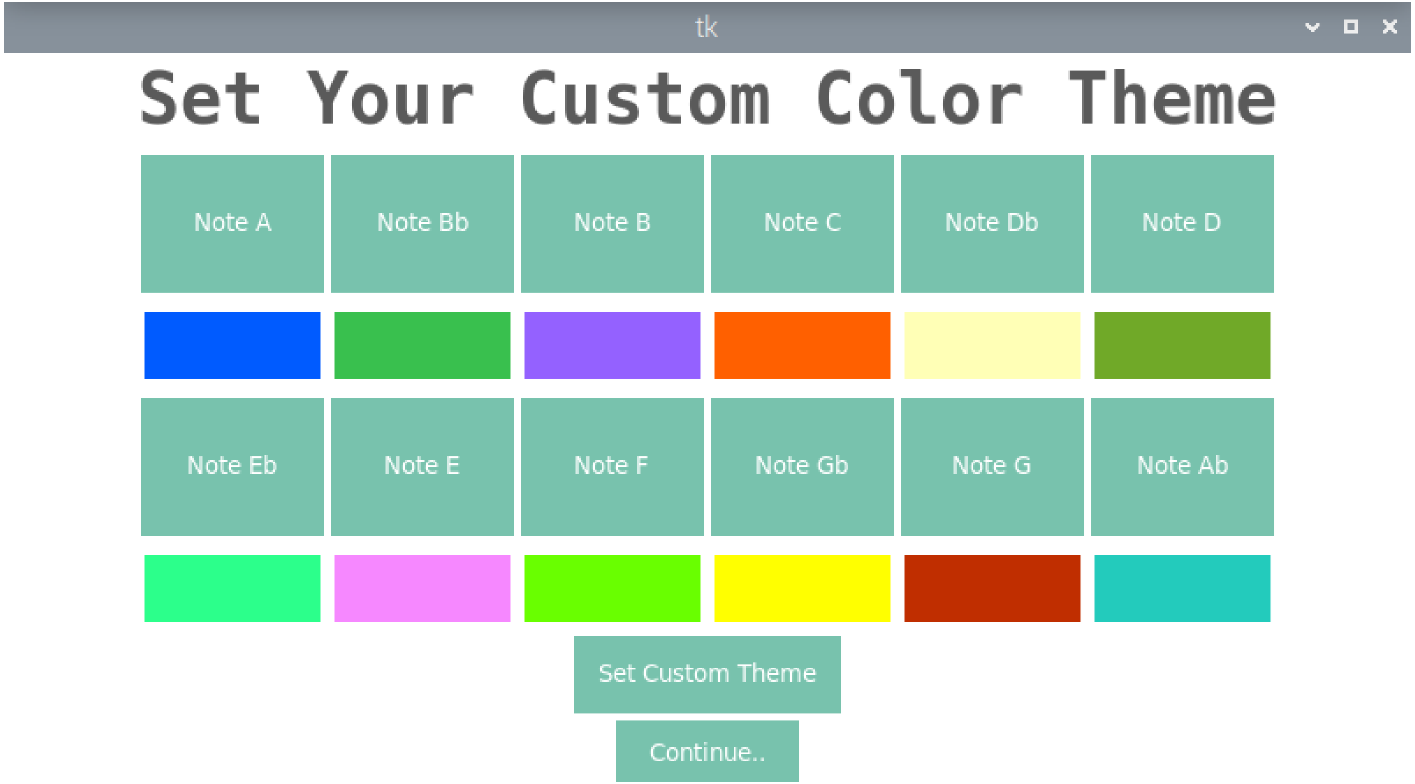
Refresh Rate

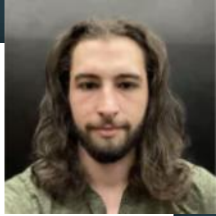
After testing we confirmed that the LEDs updated at a refresh rate of at least 5 color/brightness adjustments per second.

The example video shows 6 LED updates in the span of 1 second with a timer



Fully Customizable Note Colors

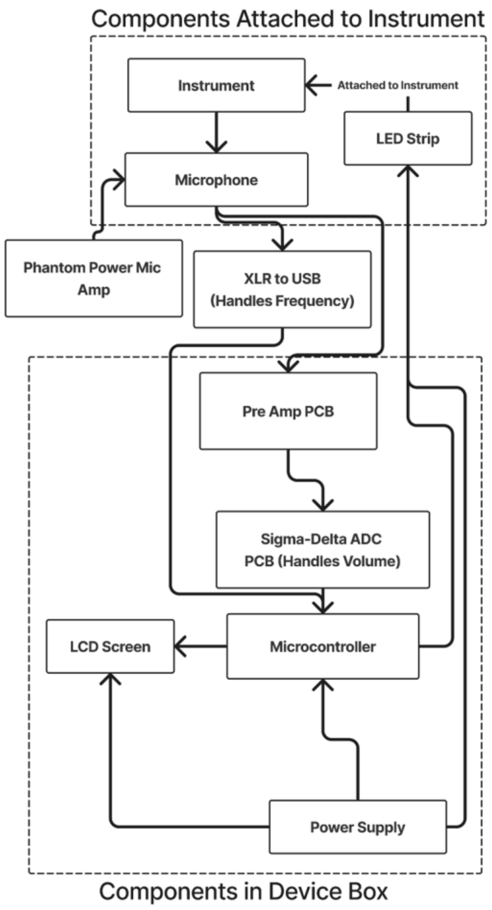




Hardware Block Diagram

We have three segments:

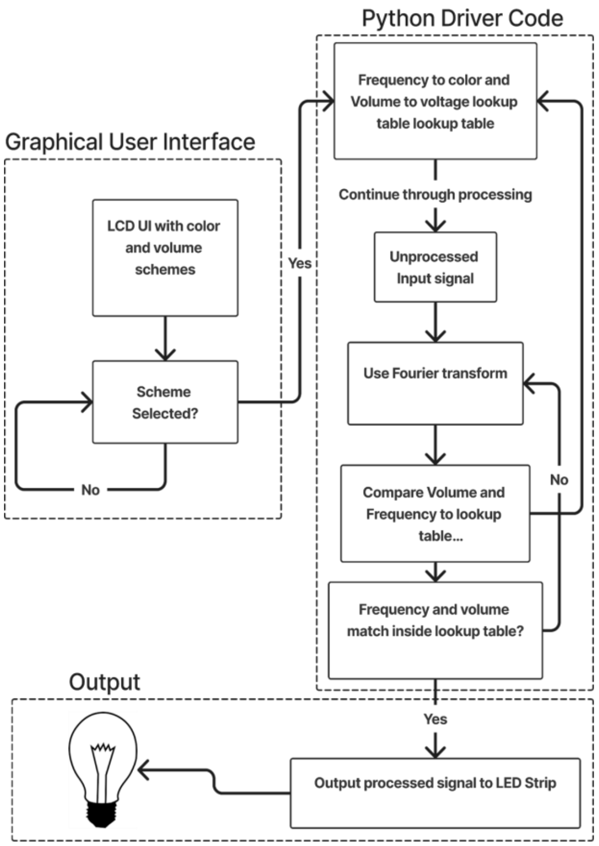
1. Components Attached to Instrument
2. Outside connections and components
3. Components in the Device Box



Software Block Diagram

We have three segments:

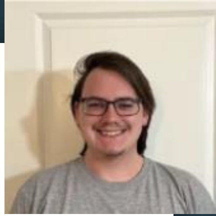
1. Python Driver COde
2. Graphical User Interface
3. Output



Component Selection

- Microcontroller
- Analog-to-Digital Converter
- LED Strips

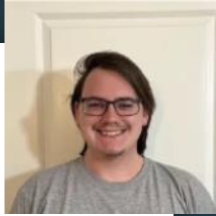
Microcontroller Comparisons



	Raspberry Pi 4	Raspberry Pi 3	Nvidia Jetson Nano
Cost	$158	$80	$298
Processor Speed	Fast	0.5x Pi 4	2x Pi 4
RAM	2 GB	1 GB	4 GB
Size	3.64”x2.76”x1.18”	3.35”x2.2”x0.8”	2.72”x1.77”x1.77”
Power Requirement	5V DC	5V DC	5V DC

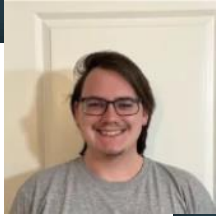
Analog-to-Digital Converter Comparisons

	Dual Slope	Flash	Pipelined	Delta-Sigma	Successive Approximation
Cost	$46.94	$154.70	$78.61	$6.66	$21.52
Size (inches)	2.07x.54	2.096x.62	0.3x0.3	0.2x0.12	0.2x0.12
Resolution	15-bit	16-bit	16-bit	16-bit	16-bit
Throughput per second	40	166,000	10,000,000	860	1,000,000
Noise	30uVpp	120uVpp	150uVpp	62.5uVpp	47.3uVpp



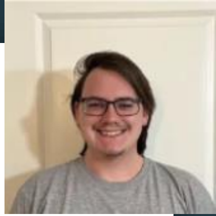
LED Comparisons

	SMD	COB
Quality of Light	Features glare	No glare Uniform light beam
Cost	~5% higher	~5% lower
Colors	Adjustable	Non-adjustable
Brightness	50-100 lumens/watt	80+ lumens/watt



Converting Music Into Color

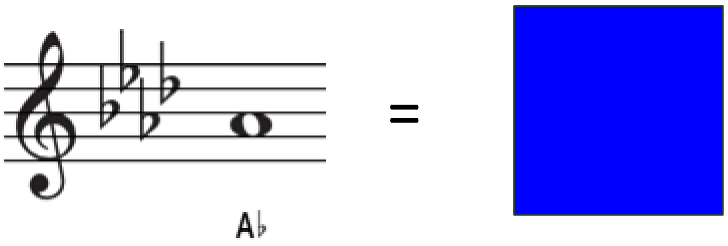
- Python program converts sound into frequency values.
- Frequency ranges assigned to various notes.
- Each note assigned a different color.
- Visual representation of Chromesthesia.



Note-to-Color Relationships

- Relationship between specific notes with specific colors
- Foundational level to be implemented
- Represents sound-to-color visualization

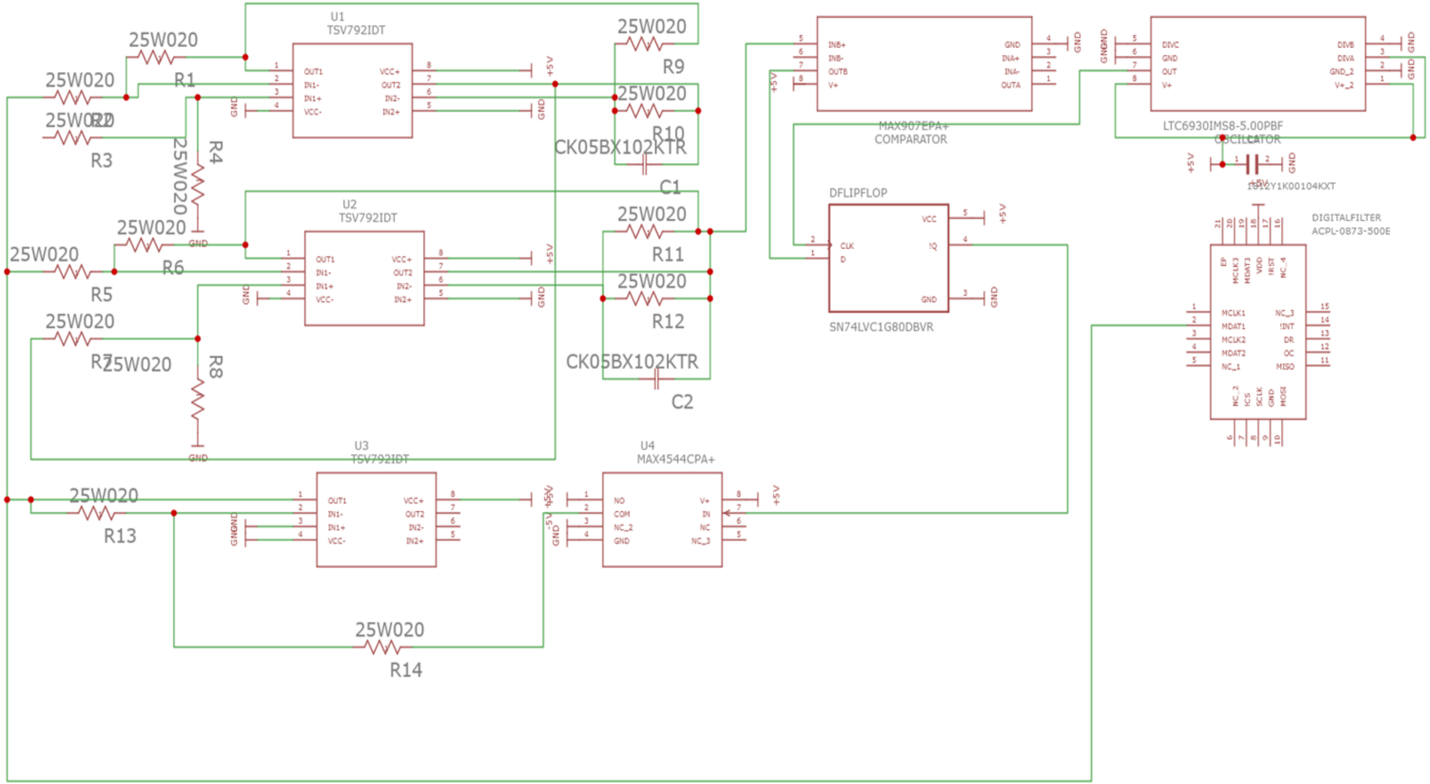
Ex. The note A flat corresponds to the color Blue



Hardware Implementation

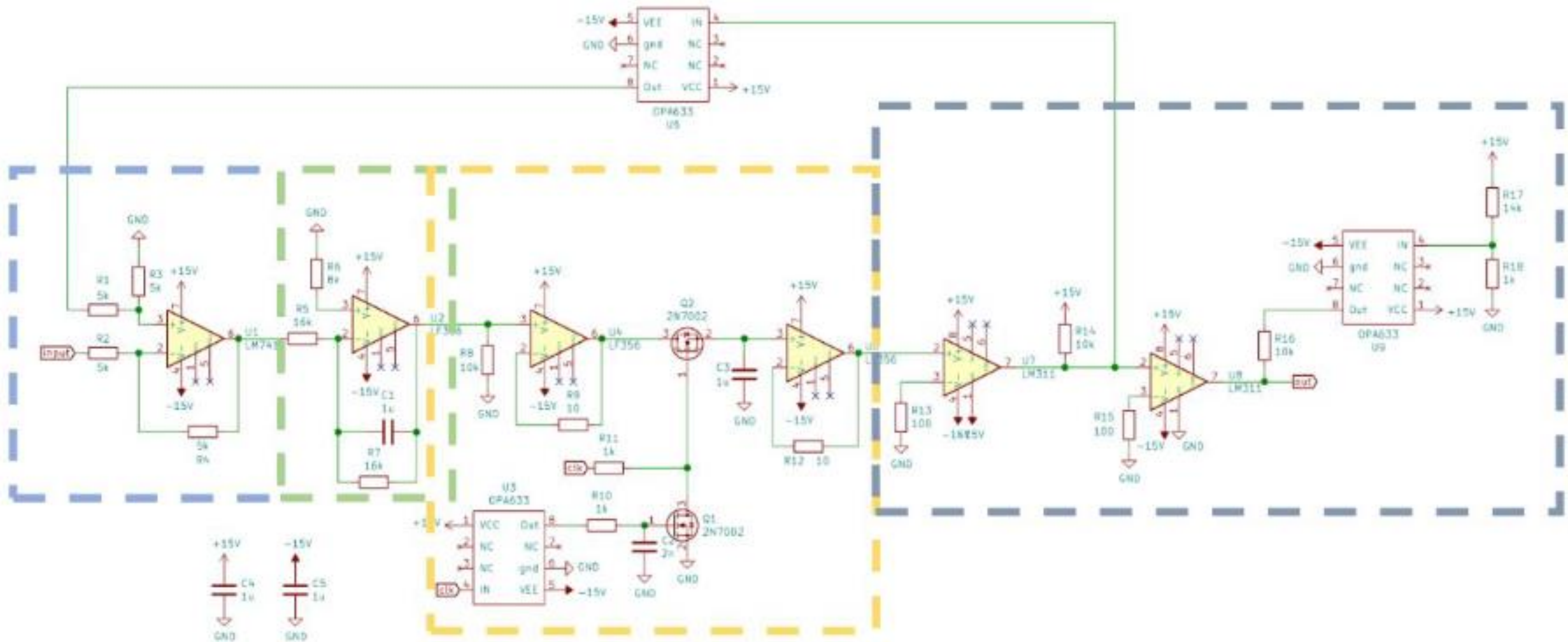
- Analog-to-Digital Converter
- Pre-amplifier
- Raspberry Pi 4 Microcontroller
- LCD screen

Analog-to-Digital Converter (Original)

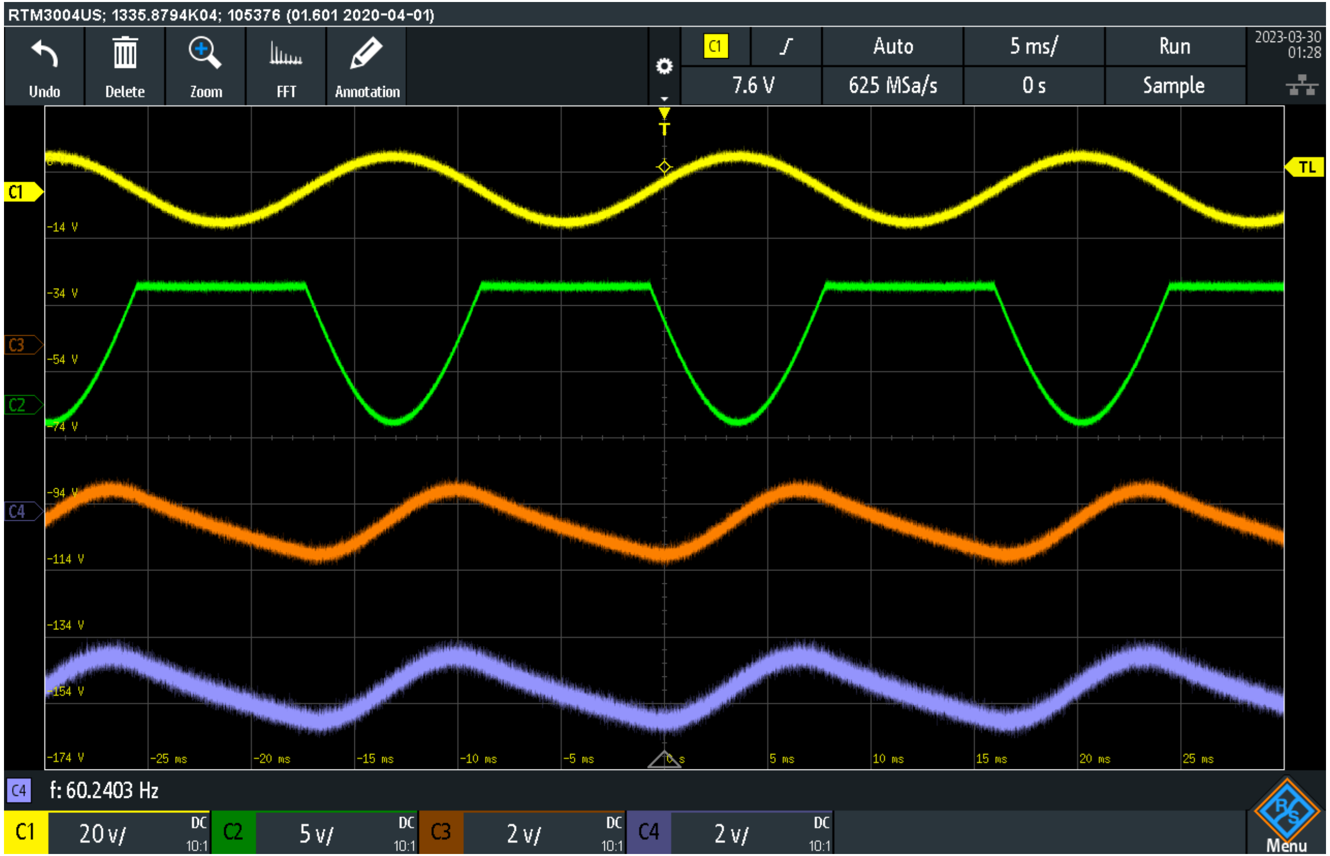


Analog-to-Digital Converter Testing (Original)

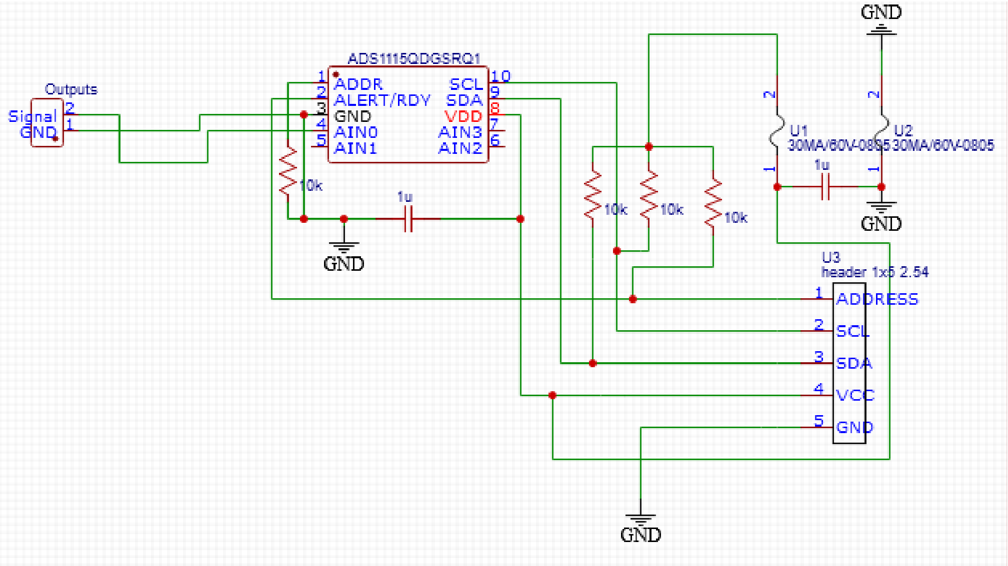
Analog-to-Digital Converter (Attempt 2)

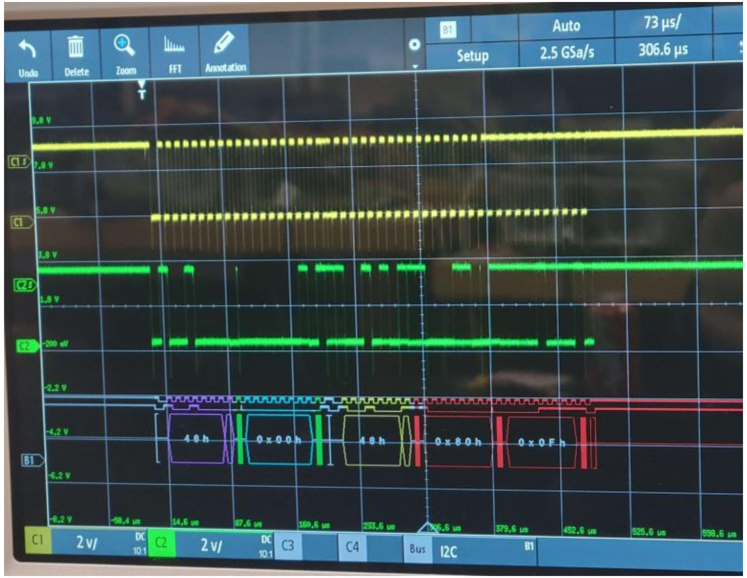


Analog-to-Digital Converter Testing (Attempt 2)

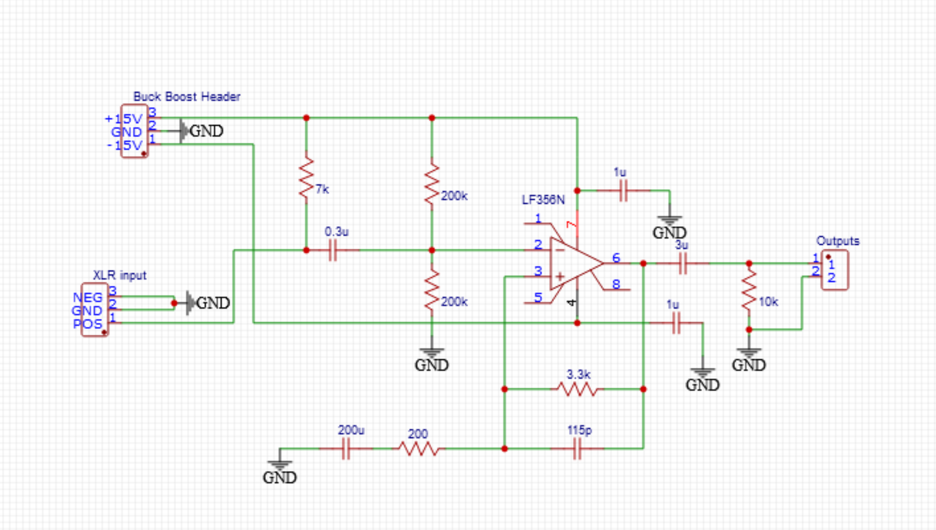


Analog-to-Digital Converter (Final)

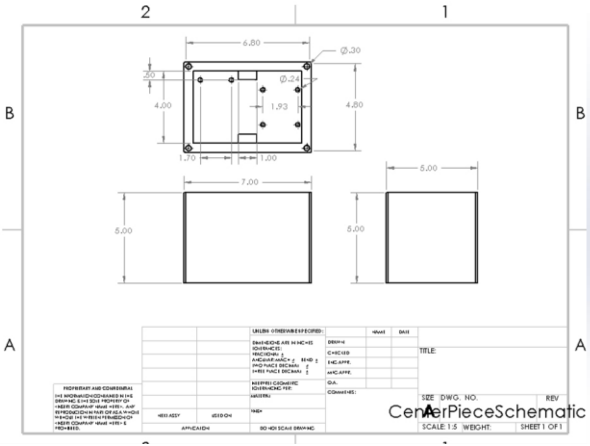


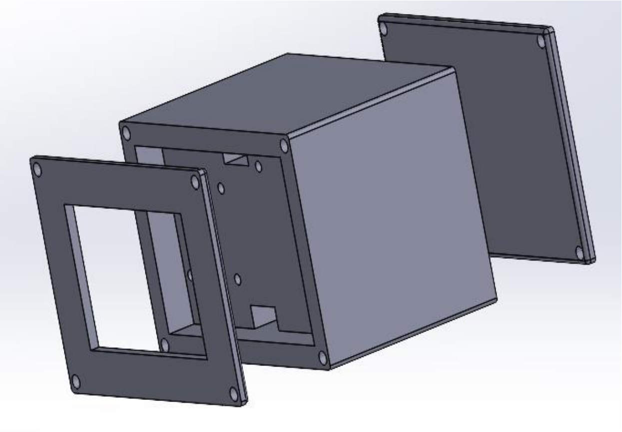


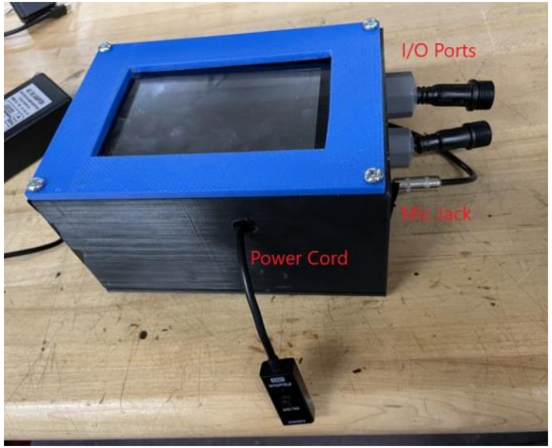
Pre-Amplifier



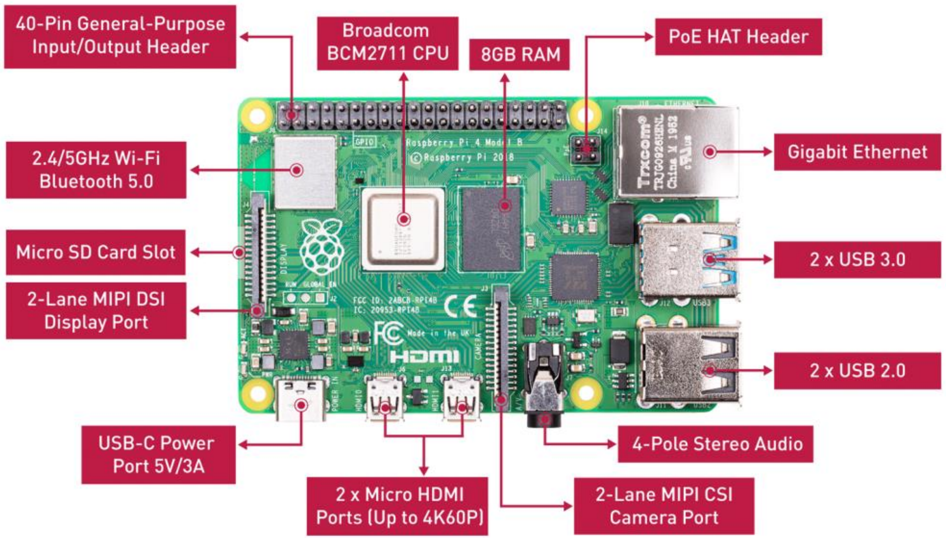
SolidWorks Design

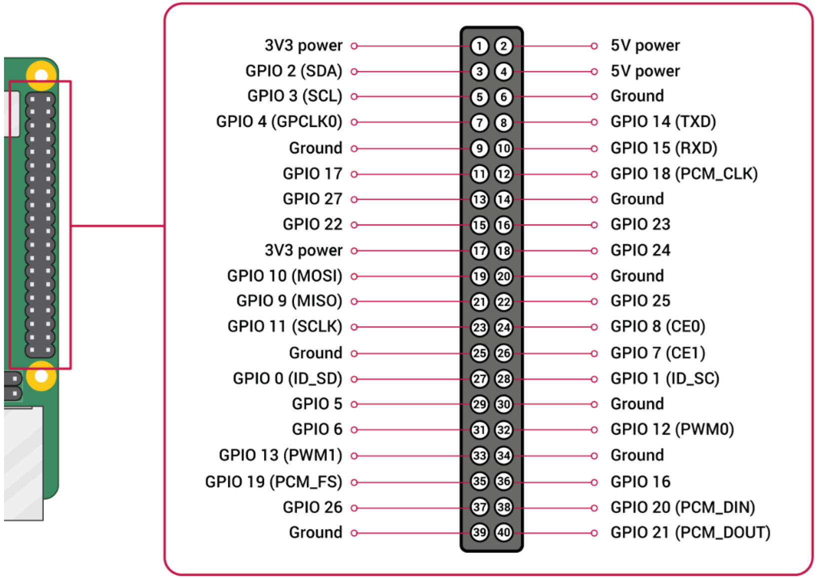


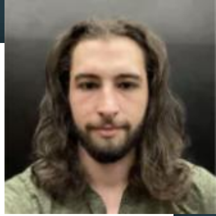




Raspberry Pi 4 Layout



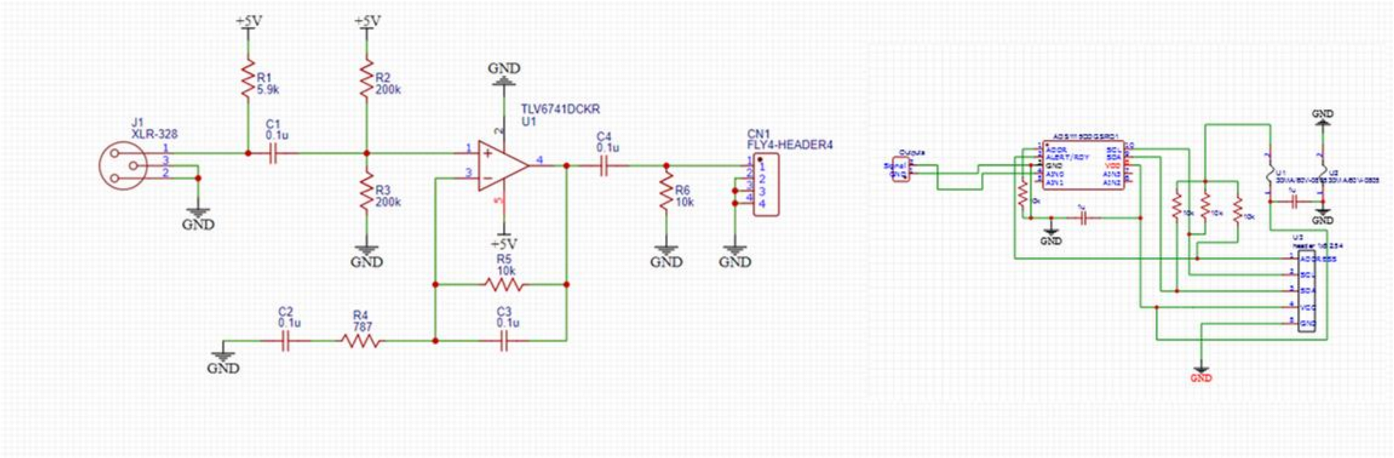




- Ports
  - 1x USB 3.0
  - USB-C Power Port
  - 2-Lane MIPI DSI Display Port

- Pins
  - 3V3 Power
  - GPIO 2 (SDA)
  - GPIO 3 (SCL)
  - GPIO 18 (PCM_CLK)
  - 2x Ground

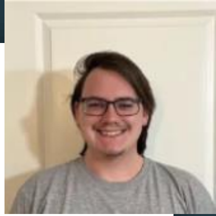
Overall Schematic



Software Implementation

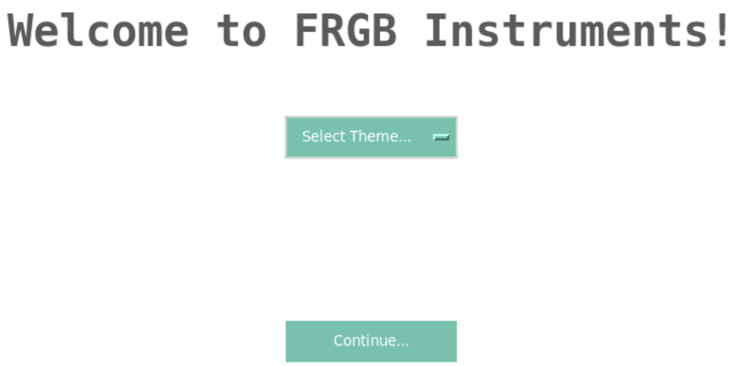
- GUI (tkinter)
- Python code
- Console output
- Software demo

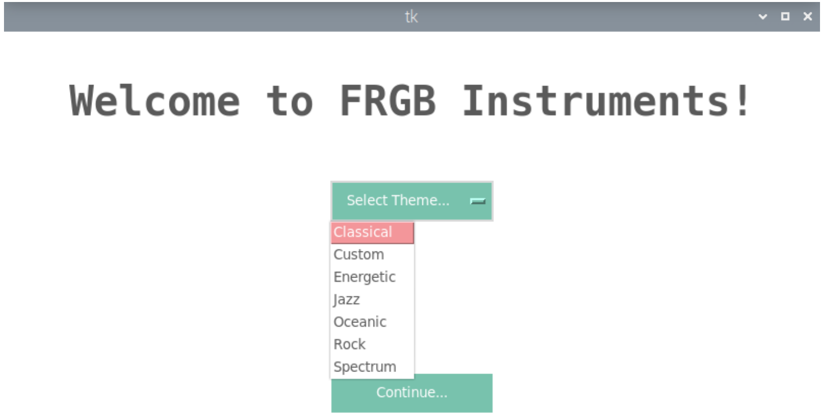
Fourier Transforms

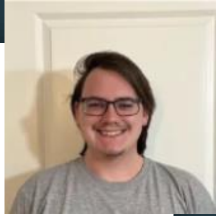


- Algorithm designed to interpret repeating patterns
  - Applied in sound, optics, electrical engineering, etc.
- Sound can be defined as the fluctuation of air pressure over time
  - FT finds the frequency of the fluctuations
- Converts a sound into a list of frequencies
  - A note is a specific frequency range
- Most sounds are made of several frequencies
  - Application uses fft.js library to implement fast Fourier Transforms
  - Fast Fourier Transform is an algorithm made for quick computer calculations

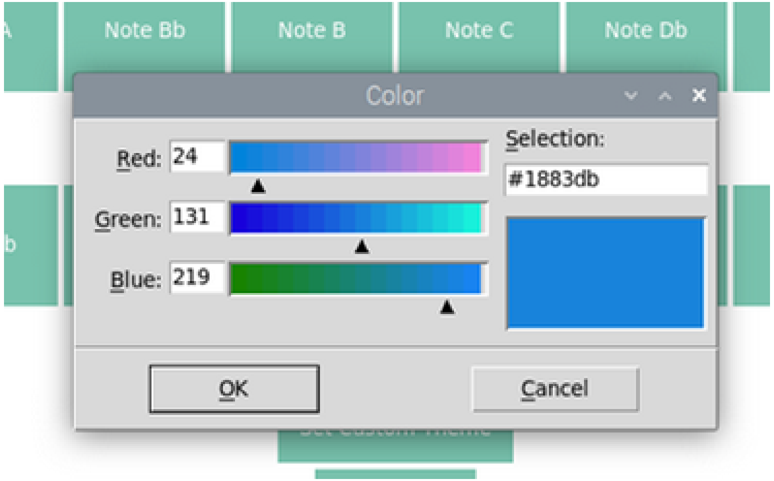
Python Gui Using tkinter



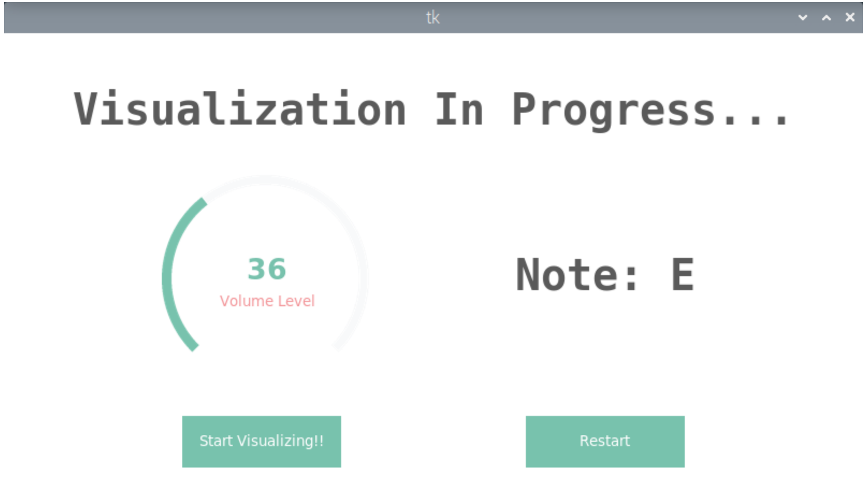


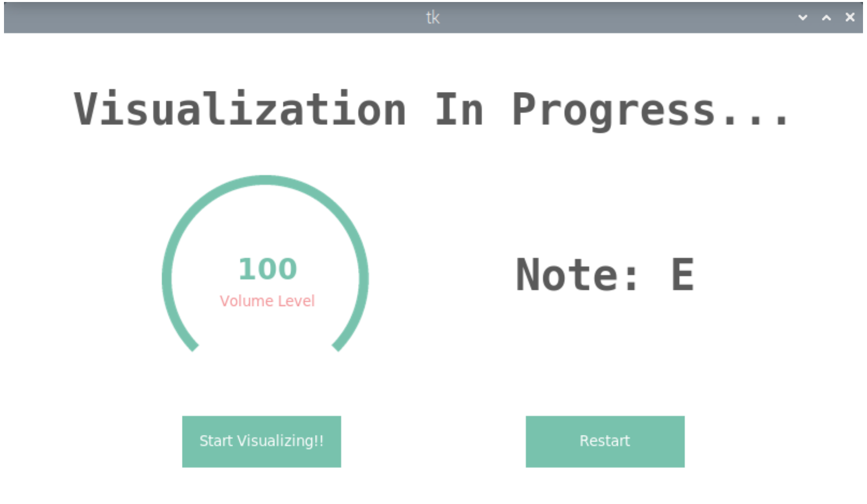


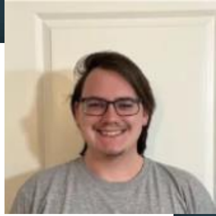
Python Gui Custom Theme Running



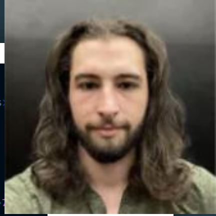
Python GUI Visualization Screen







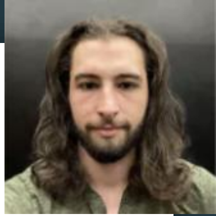
Python Code



```python
import pyaudio
import numpy as np
import board
import neopixel
import time
#Initialise a strips variable, provide the GPIO Data Pin
#utilised and the amount of LED Nodes on strip and brightness (0 to 1 value)
pixels1 = neopixel.NeoPixel(board.D18, 300, brightness=.1)
# Audio variables
CHUNK = 3200
RATE = 48000
power = 12

# Opens audio stream
p = pyaudio.PyAudio()

stream=p.open(format=pyaudio.paInt16,channels=1,rate=RATE,input=True,frames_per_buffer=CHUNK)

while True:
    # Reads the data
    data = np.frombuffer(stream.read(CHUNK, exception_on_overflow = False),dtype=np.int16)

    # Calculates the peak of the frequency
    peak = np.average(np.abs(data))*2

    # Shows the bars for amplitude
    bars = 1*int(50*peak/2**power)
    level = int(50*peak/2**power)/2
    # Calculates the frequency from with the peak ws
    data = data * np.hanning(len(data))
    fft = abs(np.fft.fft(data).real)
    fft = fft[:int(len(fft)/2)]
    freq = np.fft.fftfreq(CHUNK,1.0/RATE)
    freq = freq[:int(len(freq)/2)]
    freqPeak = freq[np.where(fft==np.max(fft))[0][0]]+1
```

```python
if(bars>2):
    #Note A
    if (107 < freqPeak <= 113) or (214 < freqPeak <= 226) or (428 < freqPeak <= 453) or (855 < freqPeak <= 906) and (bars


        pixels1.fill((255,0,0))
        print("A peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note B flat
    if (113 < freqPeak <= 120) or (226 < freqPeak <= 239) or (453 < freqPeak <= 479) or (906 < freqPeak <= 958)and (bars>
        pixels1.fill((100,50,0))
        print("Bb peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note B
    if (120 < freqPeak <= 126) or (239 < freqPeak <= 254) or (479 < freqPeak <= 508) or (958 < freqPeak <= 1015)and (bars>7):
        pixels1.fill((50,150,0))
        print("B peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note C
    if (63 < freqPeak <= 67) or (126 < freqPeak <= 134) or (254 < freqPeak <= 269) or (508 < freqPeak <= 538)and (bars>7):
        pixels1.fill((25,200,0))
        print("C peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note D flat
    if (67 < freqPeak <= 71) or (134 < freqPeak <= 142) or (269 < freqPeak <= 285) or (539 <= freqPeak <= 570)and (bars>7):
        pixels1.fill((0,255,0))
        print("Db peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note D
    if (71 < freqPeak <= 75) or (142 < freqPeak <= 150) or (285 < freqPeak <= 302) or (570 <= freqPeak <= 604)and (bars>7):
        pixels1.fill((0,150,50))
        print("D peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note E flat
    if (75 < freqPeak <= 80) or (150 < freqPeak <= 159) or (302 < freqPeak <= 320) or (604 <= freqPeak <= 640)and (bars>7):
        pixels1.fill((0,100,100))
        print("Eb peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note E
    if (80 < freqPeak <= 84) or (159 < freqPeak <= 169) or (320 < freqPeak <= 339) or (640 <= freqPeak <= 678)and (bars>7):
        pixels1.fill((0,50,200))
        print("E peak frequency: %d Hz"%freqPeak)
        #print(level)
    # Note F
    if (84 < freqPeak <= 89) or (169 < freqPeak <= 179) or (339 < freqPeak <= 359) or (678 <= freqPeak <= 718)and (bars>7):
        pixels1.fill((0,0,255))
        print("F peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note G flat
    if (89 < freqPeak <= 95) or (179 < freqPeak <= 190) or (359 < freqPeak <= 380) or (718 <= freqPeak <= 761)and (bars>7):
        pixels1.fill((50,0,150))
        print("Gb peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note G
    if (95 < freqPeak <= 100) or (190 < freqPeak <= 201) or (380 < freqPeak <= 403) or (761 <= freqPeak <= 806)and (bars>7):
        pixels1.fill((100,0,100))
        print("G peak frequency: %d Hz"%freqPeak)
        #print(level)
    #Note A flat
    if (100 < freqPeak <= 107) or (201 < freqPeak <= 214) or (403 < freqPeak <= 428) or (806 <= freqPeak <= 855)and (bars>7):
        pixels1.fill((200,0,50))
        print("Ab peak frequency: %d Hz"%freqPeak)
        #print(level)
    # Shows the peak frequency and the bars for the amplitude
else:
    pixels1.fill((0,0,0))
```

Python Code Tests (Console Output)



Single Note:

File Edit Tabs Help

```
B peak frequency: 121 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
G peak frequency: 391 Hz
```

Multiple Notes:

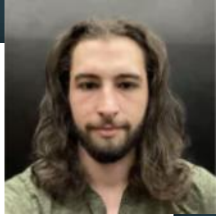
File Edit Tabs Help

```
Db peak frequency: 556 Hz
Ab peak frequency: 421 Hz
Bb peak frequency: 466 Hz
Gb peak frequency: 721 Hz
G peak frequency: 766 Hz
Gb peak frequency: 751 Hz
Eb peak frequency: 616 Hz
Db peak frequency: 541 Hz
Db peak frequency: 541 Hz
D peak frequency: 571 Hz
F peak frequency: 346 Hz
Eb peak frequency: 151 Hz
B peak frequency: 121 Hz
Eb peak frequency: 151 Hz
Ab peak frequency: 211 Hz
E peak frequency: 331 Hz
Db peak frequency: 556 Hz
Ab peak frequency: 811 Hz
A peak frequency: 871 Hz
G peak frequency: 766 Hz
E peak frequency: 646 Hz
Gb peak frequency: 721 Hz
Gb peak frequency: 721 Hz
```

Challenges Faced With Hardware

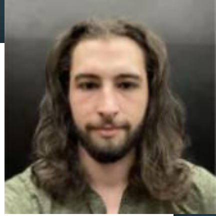
Problem	Solution
Aimed too highly for course PCB requirements.	Building an ADC from scratch proved far too difficult and time consuming when paired with the rest of the project.
Shipping delays with PCB components.	Paid extra for 4-6 day shipping.
Component failure.	Purchased multiple backups of each component.
Issues with understanding component datasheets	Experience from time in the lab testing components as well as research
Finding recommended parts	Using replacement parts

Challenges Faced With Software



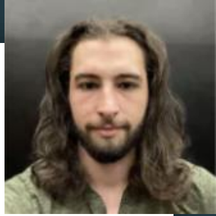
Problem	Solution
Corrupt Python Libraries on Raspberry Pi that stopped the program from running	A clean Raspberry Pi OS install (Seemed like a one off corruption)
How to Implement a GUI with Python	Utilized the Tkinter and ttk bootstrap Python libraries to implement GUI with widgets
Sample Rate	Trial and error until the correct gain and sample rate values were found.
Debugging Issues with Raspberry Pi	Research and Trial and Error
Inability to have more than one person work on code while away	More time spent together in Senior Design Lab and phone calls

Work Distribution



Name	Stefan	Anja	Leith	Carson
Software Design			P	P
Hardware Design	P	P		
ADC	S	P		
Pre-Amplifier	P	S		
RGB Integration			P	S
GUI			S	P

Legend:
P = Primary
S = Secondary



Future implementations

- Note Intervals (i.e perfect fifth).
- Bluetooth module and lights.
- Merging software application with physical product.
- Training mode
- Polyphonics

Any Questions?