HARMONY SYNTHESIZER GUITAR PEDAL

By

Danielle Lange
Madhav Khirwar

Final Report for ECE 445, Senior Design, Fall 2021
TA: Feiyu

8 December 2021
Project No. 28

Abstract

The product we developed over the course of the past semester is a harmony synthesizer guitar pedal that is applicable in a variety of musical contexts. Our product saves a guitarist time by simplifying orchestration/arrangement since it allows the guitarist to record/perform synth parts alongside guitar parts. The guitarist has the option to use the product to simply double their guitar part on the synth in unison, or to use the pedal to provide harmonic context to the melody they are playing. It also allows them to plug the pedal into a DAW/external synthesizer for simplified recording and custom timbres.

Our prototype was easy to add to an existing guitar pedal chain and was able to successfully provide both one-to-one doubling as well as harmony to a guitarist.

Contents

1. Introduction

1.1 Problem

A lot of guitarists these days, especially in the metal/indie scene, like to double/harmonize their guitar parts with synthesizer parts overlayed on top [2]. This is usually done note-for-note, and usually wastes a lot of effort as the guitarist must re-record the synth parts over the guitar tracks. There is a lot of wasted effort that is directly proportional to how complex/fast the guitar parts are.

1.2 Solution

This product is meant to act as a device (guitar pedal) that adds musical context/layering on the notes played on a guitar. The guitarist will benefit from this product in that he/she can create more specialized sounds by blending different notes and/or chords. This would make it easier for the musician to achieve the specific sound/note that they desire.

Our product is marketable because the cost will save a lot of time for the artist when they are tracking guitar parts and doubling them with synths, as well as cut down on the amount of stage gear they need when performing live. Our approach is to use a DSP-capable microprocessor (such as the MK20DX256VLH7 on the Teensy 3.2) to implement a novel kind of guitar pedal – one that will add electronically synthesized harmonies to an analog guitar note played by the guitarist. This will be in the form factor of a normal pedal, and the knobs will give the guitarist the option to select what kinds of harmonies they want to overlay (Major 5th, Major triad, 7th, etc).

Our chassis for the pedal is of dimensions (14.5 cm, 12.1cm, 3.9cm), and has 2 female ¼ inch connectors, one MIDI out, and one 9V DC power input. We wanted the dimensions to be such that it fits on a standard guitar pedal board and is compatible with the rest of the guitarist's signal chain. In terms of controls, we want the guitarist to be able to specify the harmony they would like to generate as well as the type of waveform (from square, sawtooth, or sinusoidal). The electric guitar's output will be connected to the input female ¼ inch connector on the pedalboard. The guitarist can use the harmony selector switches on the pedal to select what harmonies to generate as MIDI notes, and the waveform selector switches to select what waveform to generate. If the guitarist wants to use their own external synth with a more elaborate/customized waveform generator or signal chain, they can route the MIDI notes generated to the external synth. The output of the pedal would then be connected to either an amplifier or the rest of the guitarist's pedal chain. The guitar signal itself is mixed as-is with the generated harmonies.

1.3 Requirements

We listed the high-level requirements as defined in our Final Design Document.

- Can resolve a frequency in the range 50-1500 Hz to the closest note in 12-tone equal temperament tuning system with A at 440 Hz. Error tolerance of +/- 5%.
- ADC/DAC should be at a sample rate of at least 44.1 kHz and at least 16-bit resolution [8]. This is because CD quality audio has 16-bit resolution and is sampled at 44.1 kHz.

- Makes overall music production and performance significantly more convenient for the guitarist.
- Total end-to-end latency should be under 10 milliseconds. For guitar, this is when guitarists can 'feel' the performance getting sluggish even though listeners cannot tell the difference [3].

Frequency resolution is important because we want our product to be able to work for all guitar notes. We have the ADC/DAC requirement because it is a standard for CD audio quality. At a lower bit resolution, the output of the guitar could sound noisy and not provide the same quality. We want to make overall product convenient for consumers. This aligns with our main goal to solve the problem of being able to harmonize and synthesis simultaneously. The Latency requirement is necessary to our product because at 10ms a guitarist may notice the delay between played and hearing a note [4]. Without this requirement it would be hard for the guitarist to play on beat.

1.4 Subsystem List

- The **Power Subsystem** is responsible for delivering 9v DC power to the input buffer subsystem, output buffer subsystem, and the Teensy. The Power Subsystem is takes input from the power supply.
- The **Input Buffer Subsystem** is responsible for dividing the peak-to-peak voltage of the guitar signal and biasing it to 0.6v so in can fall with range of voltage required by the Teensy analog input pins. The guitar signal is the input of the Input Buffer Subsystem, and the on-board ADC Subsystem is where the output goes.
- The **Output Buffer Subsystem** is responsible for reamplifying the signal, inversing the input buffers modification to the signal. The output buffer takes input from the DAC in the Teensy and then outputs to the speaker.
- The **Combinational Subsystem** is responsible for combining the original guitar signal after ADC and the signal after synthesis.
- The **ADC/DAC Subsystem** is responsible for digitizing the buffered guitar signal input for digital signal processing, and then converting the processed output back to an analog signal to be buffered by the output buffer.
- The **Frequency Analyzer Subsystem** is responsible for recognizing what note the guitarist has played from the signal after A/D conversion.
- The **Synthesizer Subsystem** is responsible for outputting the user-selected harmony as a user-selected waveform based on the note recognized by the frequency analyzer.

2 Design

2.1 Block Diagram

2.2 Physical Design

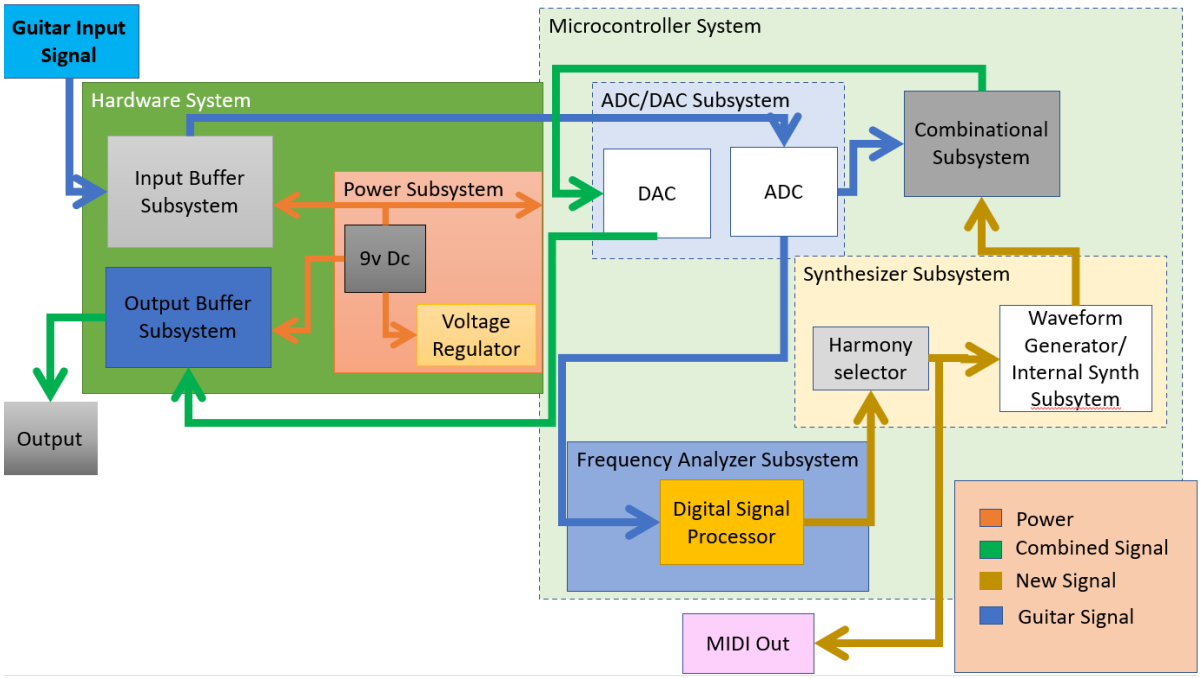

Figure 1: Block Diagram for our product broken down into systems and subsystems.

Figure 2: From left to right on the pedal diagram above, there is the top panel of the guitar pedal. The switches on the top turn individual harmonies on/off. The other three switches from top to bottom control the waveform, the major/minor quality of the 3rd and 7th intervals, and toggle between True Bypass and discrete/continuous note recognition. The side panel of the guitar has the ¼ inch input and output as well as the barrel 9V center negative jack.

2.3 Hardware

2.3.1 Input Buffer

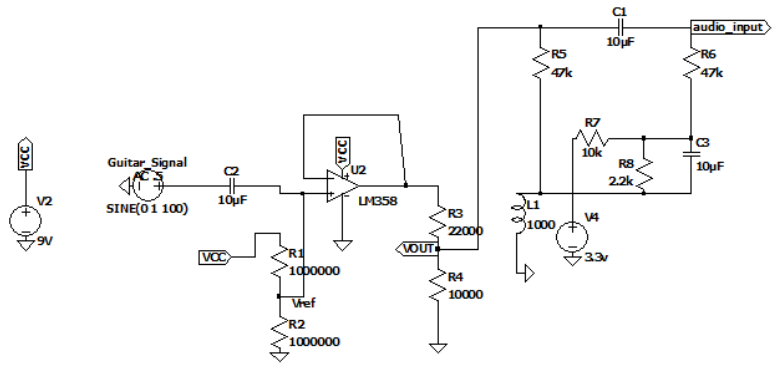

Figure 3: Input buffer schematic.

Note that in the Figure 3 L1 and V4 do not exist on the PCB they are the on-board circuitry of the Teensy inputs they are there for clarity of design. Also note, the audio_input is the output of the input buffer. The input signal to the ADC is attenuated by roughly a factor of 3, since this stage is powered directly by the 9v power supply, it has roughly 9V peak-to-peak headroom. R3 and R4 will achieve this attenuation.

$$\frac{10}{32}=1+\frac{10k}{22k+10k} \tag{1.0}$$

The 9v power supply is divided in half by R1 and R2 in Figure 3. The purpose is to bias the guitar signal because the input to Teensy cannot read negative voltages. The circuit to the right of VOUT in Figure 3 bias's the signal to half of the reference voltage of the Teensy input which is 1.2v [1]. The original design only includes the right half of the circuit ending at VOUT. Since we changed the audio board to be a DAC board instead the output of the input buffer will feed directly into Teensy instead of the Teensy audio board. The difference with this is that the Teensy audio board has an input voltage range of 0-3.3v whereas the Teensy can only take a range of 0-1.2v. We decided to keep the original circuit since it is consistent with our output buffer logic and the guitar signal still needs to be attenuated to make sure it fits in the acceptable range of the Teensy analog voltage range.

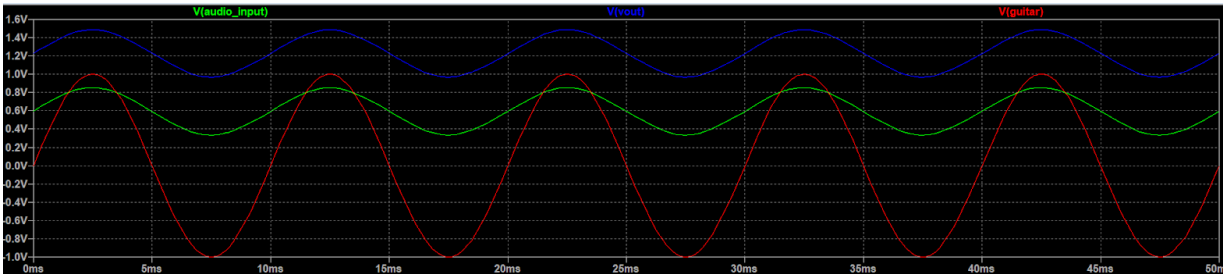

Figure 4: Output of original input buffer vs whole input buffer

Note that the Figure 4 uses a guitar signal of 2v peak to peak which is well above the maximum guitar note we is possible [12]. The result is a waveform that is almost exactly a third of the original peak to peak voltage. The new part of the design will always bias the signal to roughly 0.6v.

$$0.595v = 3.3\frac{22k}{22k+10k} \quad (1.1)$$

As seen in Equation (1.1) the attenuation is because of resistor R7 and R8 and the 3.3v output pin from Teensy. This is useful for putting the signal exactly between the range of the Teensy voltages so that there is the most room for the signal.
The alternative design to this subsystem that we were considering would include only R1, R2, C2, LM358, R3, and R4 which is the left half of the circuit. This design would work with using the Teensy audio board instead of the DAC board as we originally intended.

2.3.2 Output Buffer

The output signal from the DAC needs to be amplified by a factor of 3. This will reverse the signal attenuation from the input buffer. This is important because with a smaller voltage the sound will change its strength and have more room for noise. The outputs buffers schematic can be seen in Figure 5.

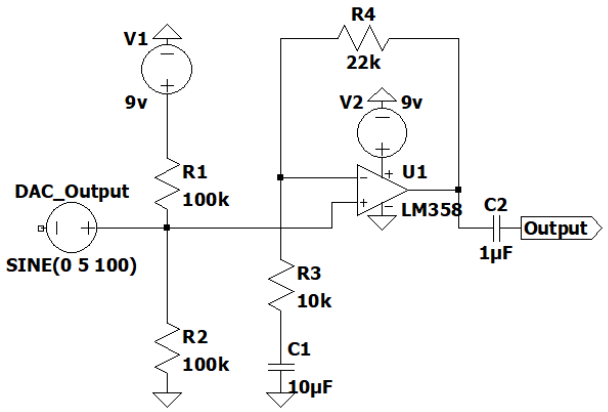

Figure 5: Output buffer schematic

$$\frac{10}{32} = 1 + \frac{22k}{10k} \quad (1.2)$$

The gain of this buffer is represented by Equation (1.2). This is the exact opposite attenuation from the input buffer seen in Equation (1.0) which is what we wanted.

2.3.3 Power

This subsystem's purpose is to supply 9v to all the different parts of the design, as well as to ensure that the Teensy receives 5v. Since the power supply needs to be compatible with the rest of the guitarist's pedalboard, it will take a 9V dc input. However, the Teensy microcontroller operates on a maximum of 6v, so we will use a LM7805 voltage regulator ic to convert from 9V to 5V [6]. Since the Teensy operates on about 60mA of current, and the voltage drop is 4V across the voltage regulator, this subsystem will dissipate about 0.25 Watts as heat [11]. This should not make it appreciably hotter than room temperature and should still be quite safe to the circuit.
The digital signal processing that occurs on the Teensy board will occur in 3.3v. However, Teensy has on-board voltage regulators that will step 5V down to 3.3V for DSP operations [1].

An alternative design would be to use a battery powered power supply. The problem with the alternative is that batteries die after some time and will require replacing and spending more

money. Also, a battery provided less constant voltage because it weakens as it nears the end of its battery life.

2.3.3 Analog to Digital Convertor

This subsystem manages converting the signal from analog to digital which is critical to being able to harmonize the signals. ADC was implemented natively on the Teensy's analog input pins, which was a result of excluding the Teensy audio shield from our original design. The MK20DX256VLH7 on the Teensy board is capable of ADC at 16-bit depth sampled at 44.1 kHz. The pinout of the Teensy audio board can be seen in Figure 6.

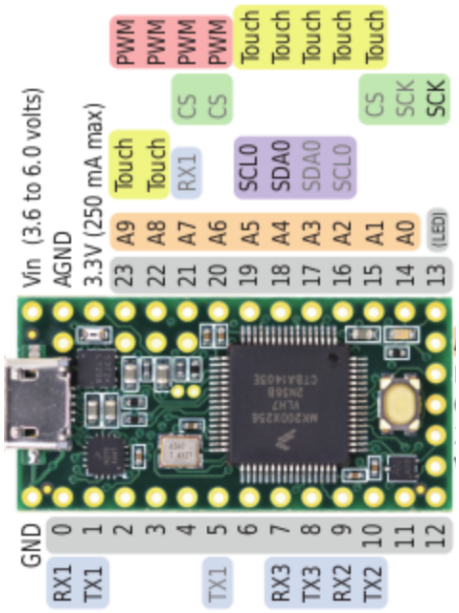

Figure 6: The Analog to Digital Conversion is handled by the Teensy when input is received at pin A2 (16). This is where the guitar signal is received after the input buffer.

2.3.3 Digital to Analog Convertor

This subsystem manages converting the signal from digital to analog after the addition of the harmony notes. Since we are aiming for the highest harmony note to be an octave (double the frequency of the input), the constraints and requirements of the DAC are higher. This was implemented on a custom DAC board we implemented for the Teensy based on the PT8211 DAC chip, which is capable of 16-bit DAC at 44.1 kHz. The connections to the Teensy 3.2 can be seen in Figure 7.

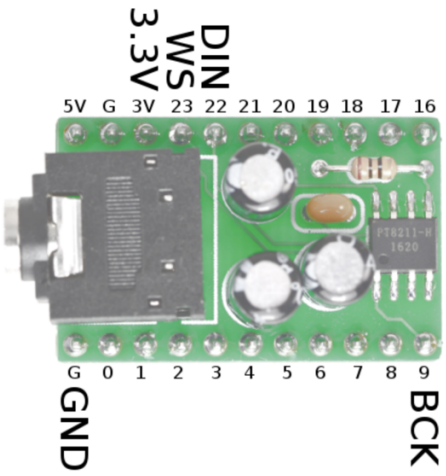

Figure 7: The DAC circuit is connected to the Teensy 3.2 as described in this image

2.4 Software

The general software for our product can be seen in Figure 8.

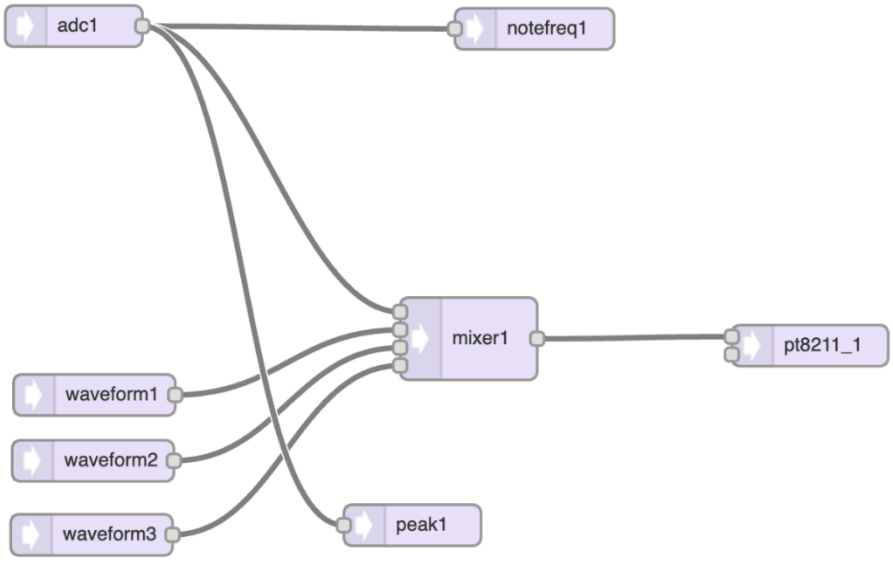

Figure 8: Graphic Representation of the DSP Signal chain as seen in Audio Design Tool [2]

2.4.1 Frequency Analyzer Subsystem

This subsystem is meant to recognize the note that the guitarist has played after Analog to Digital Conversion. This was done by mapping the frequency of the played sound to the closest note in the 12-tone equal temperament, A at 440 Hz system. In order to deduce the note that has been played, we used the YIN algorithm to estimate fundamental frequency.

As described by Cheveigne and Kawahara [12], this is a 6-step autocorrelation in time-based algorithm that estimates the fundamental frequency of a monophonic signal. In the above figure, this is the component labelled 'note2freq1'. The DSP system also recognizes a new note by recognizing a peak in the guitar signal, so as to only read notes when a pick stroke or tap is detected, and this is implemented in the component labelled 'peak1' above. The decay of a guitar not can be seen in Figure 9.

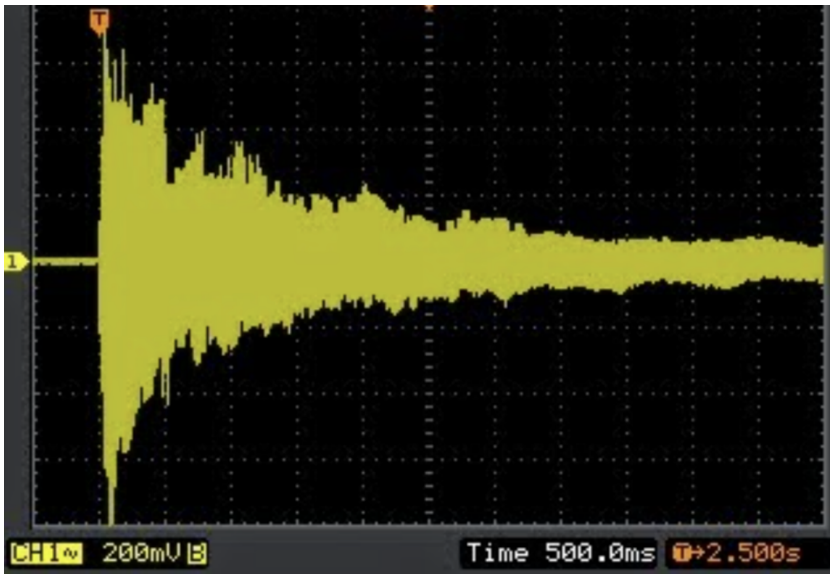

Figure 9: Decay of a single picked guitar note.

Once a note is played, it then needs to be mapped to the correct frequency according to the 12 tone equal temperament system with A at 440 Hz. The following are the frequencies of the 12 notes on the guitar in the lowest frequencies [11]:

Table 2 Frequencies of the Lowest 12 Notes on an Electric Guitar

Note	Frequency in Hz
E	82.407
F	87.307
F#	92.499
G	97.999
G#	103.826
A	110
A#	116.541
B	123.471
C	130.813
C#	138.591
D	146.832

The neck of a 24-fret guitar, as shown below, spans 4 octaves. Since an octave above a note is the frequency of the note multiplied by 2. Thus, to calculate the set of notes that are possible on the guitar neck, we took the above notes and doubled them for each available octave. This gave

us a set of 49 distinct notes possible on a guitar. At first glance this may seem incorrect as there are 24 ways to fret each of the 6 strings. However, many notes are repeated on the guitar neck. For reference see a guitar in Figure 9.

Figure 9: A 24-fret Ibanez Electric Guitar

We thus stored all 49 possible notes on the Teensy in a sorted array of floats, as seen in Appendix C:

```
float guitar_notes[GUITAR_NOTE_COUNT] = {82.41, 87.31, 92.50, 98.00, 103.8, 110.0, 116.5, 123.5,
 130.8, 138.6, 146.8, 155.6, 164.8, 174.6, 185.0, 196.0, 207.7, 220.0, 233.1, 246.9,
  261.6, 277.2, 293.7, 311.1, 329.6, 349.2, 370.0, 392.0, 415.3, 440.0, 466.2, 493.9,
 523.3, 554.4, 587.3, 622.3, 659.3, 698.5, 740.0, 784.0, 830.6, 880.0, 932.3, 987.8,
  1047,  1109,  1175,  1245 , 1319};
```

Then, in order to 'freeze' the output of the YIN algorithm to the 'correct' note, we perform binary search through the sorted array with the YIN output as the target term. Numerically, the closest term is then assigned as the played note. Binary search algorithm as in Appendix C:

```
float bin_search(float *arr, int start_idx, int end_idx, float search_val) {

   if( start_idx == end_idx )
      return arr[start_idx] <= search_val ? arr[start_idx] : arr[start_idx];

   int mid_idx = start_idx + (end_idx - start_idx) / 2;

   if( search_val < arr[mid_idx] )
      return bin_search( arr, start_idx, mid_idx, search_val );

   float ret = bin_search( arr, mid_idx+1, end_idx, search_val );
   return ret == 0 ? mid_idx : ret;
}
```

2.4.2 Synthesizer Subsystem

Once the correct note has been calculated by the frequency analyzer subsystem, the Teensy reads the inputs at the switches via its digital input pins. The harmony notes that are selected are calculated with intervallic multipliers. The following section of code sets the frequency of each harmony note based on whether or not the interval is selected by the user, as in Appendix C:

```
if(octave_on){waveform1.frequency(note * 0.5);}
else{waveform1.frequency(0);}

if(fifth_on){waveform2.frequency(note * 1.49831);}
else{waveform2.frequency(0);}

if(third_on){
  if(!major_minor){
    waveform3.frequency(note * 1.18921);
    waveform4.frequency(note * 1.782);
    } // minor
  else{
    waveform3.frequency(note * 1.25992);
    waveform4.frequency(note * 1.888);
    } //major
  }
 else{waveform3.frequency(0);}
```

This note information is also passed onto the MIDI output that we were unfortunately unable to test, due to shipping troubles with our MIDI connector package. The MIDI port connection can be seen in Figure 10.

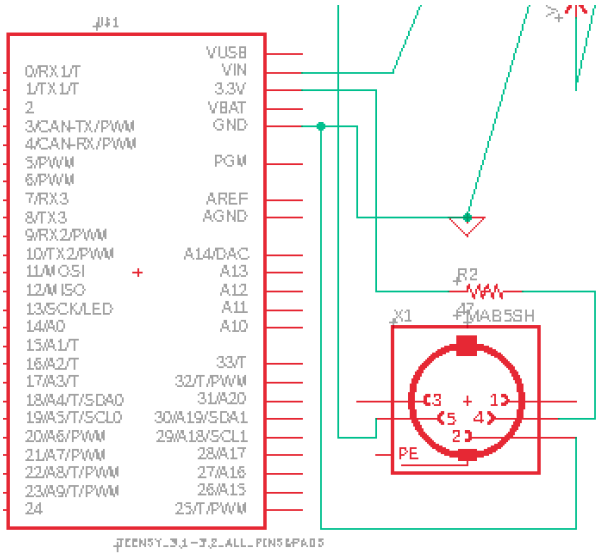

Figure 10: The MIDI port driven by the Teensy

Finally, the note information is also passed onto the internal waveform generators for each interval. The switches connected to the Teensy's digital inputs define the selected harmony, and the waveforms are generated accordingly and passed onto the DAC subsystem.

3. Design Verification

3.1 Latency

The Latency of the end-to-end system must be less than 10ms. To verify this is simple. We compared the oscilloscope readings from one end of the system to the other. The difference between the peaks of the signal will show the total latency. The reason for this is that, as described in an article by A. Swanson[4], 10ms seems to be the latency around which a performer can 'feel' a lag even though the audience can't perceive one.

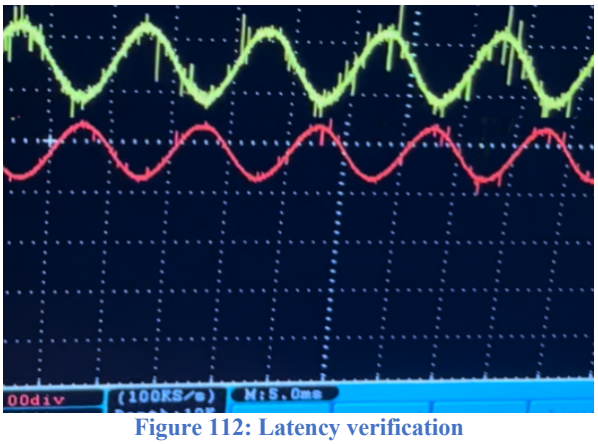

Figure 112: Latency verification

Figure 11 shows that the peaks of each waveform are 5ms apart in time. A problem with this is not knowing if it is several 5ms apart. This can be proved by showing the delay using a more irregular waveform the problem with this is guitar signals too uniform. Our solution was to input a high amount of noise from our computer as the input guitar signal.

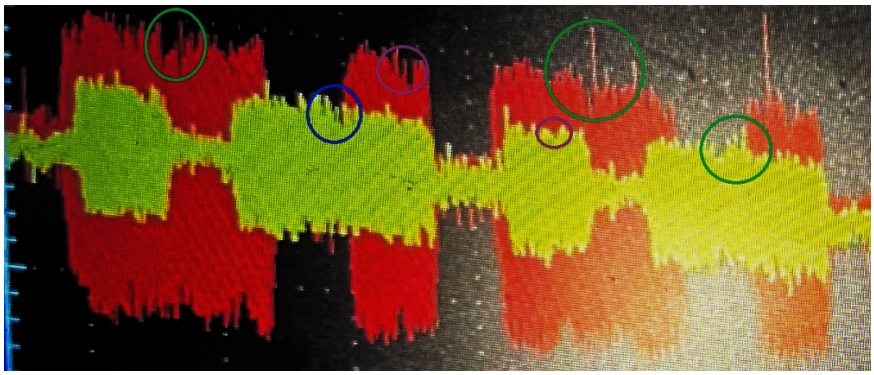

Figure 123: Latency second latency verification

Figure 12 is again 5ms across each horizontal grid line. Note that the yellow signal is the output signal where the red is the original guitar signal. We compared the yellow signal with the waveform of the red signal 5ms before. While very noisy there is a pattern between the lowest dips and the highest peaks of the original and final signal.

3.2 Convenience

An important verification for our project is verifying that it is convenient for the guitarist to use. Part of the convenience factor is that we used switches instead of rotary potentiometers. The reason being is the rotary potentiometers need to be spun which will more than likely require turning it by hand. Verification for this is we played the guitar and made sure the switches and button could be operated with our foot.

3.3 Frequency Range

Need product to resolve a frequency in the range 50-1500 Hz to the closest note in 12-tone equal temperament tuning system with A at 440 Hz [12]. With an error tolerance of +/- 5%. For this verification we demonstrated playing guitar notes with the binary search on. The binary search 12-tone equal temperament is verified by design in the code used for Teensy. We chose to test within this frequency range since it encapsulates the lowest and highest notes that can be produced on a standard electric guitar with 6 strings and 24 frets tuned in standard EADGBE tuning. The actual range of the frequencies is 82-1320 Hz.

3.4 ADC/DAC Sample Rate and Bitrate

ADC/DAC should be at a sample rate of at least 44.1 kHz and at least 16-bit resolution [7]. This is because CD quality audio has 16-bit resolution and is sampled at 44.1 kHz. This is verified by design. This is because the DAC board and the Teensy 3.2 ADC are both rated for a 16-bit resolution and sampling rates of 44.1kHz [2].

4. Costs

4.1 Parts

The actual cost for all the parts used in our final project is 58.69 which is amazing. This is an extremely cheap for what our product does and very exciting. The part-by-part calculation of this is outlined in Table 2.

Table 2 Parts Costs

Part	Manufacturer	Quantity	Retail Cost ($/Item)	Bulk Purchase Cost ($/Item)	Actual Cost ($/Item)
Teensy 3.2	PJRC	1	19.80	13.00	19.80
Break Away Headers	SparkFun Electronics	1	1.50	1.00	1.50
DPDT Switches	SparkFun Electronics	6	0.75	0.25	4.00
Footswitch	SparkFun Electronics	1	3.00	1.00	3.00
PT8211 DAC	Princeton Tech	1	2.65	1.50	2.65
LM358-N	Texas Instruments	2	1.57	0.05	0.20
L7805	SparkFun Electronics	1	0.95	0.25	0.95
Power Connector	SparkFun Electronics	1	0.75	0.25	0.75
Audio Jack	Markertek	4	2.25	0.60	2.25
PCB Board	PCBWay	1	5.65	2.00	5.65
Power Supply Connector	D'Addario Accessories	1	9.99	5.00	9.99
Resistors	Panasonic Electronic Components	13	0.05	0.01	Free
Capacitors	Texas Instruments	4	0.31	0.02	Free
Total			**61.00**	**28.10**	**58.69**

4.2 Labor

The expected cost per person assumes an average of the reported Computer and Electrical Engineering graduating salaries from 2019-2020. They are sequentially 110,978 and 76,129 making the average 87,637 a year [1].

$$Labor\ cost = \text{ideal salary (hourly rate)} \times \text{actual hours spent} \times 2.5 \quad (4)$$

Using the Equation (4) and with a combined total labor hour estimate of 150 hours we would calculate labor cost to be 32,863,875$. The schedule for this work is outlined in Appendix B in Table 6.

5. Conclusion

5.1 Accomplishments

This project was very successful. We wanted to make it easy for guitarist to harmonize and synthesize their guitar notes without delay. This was able to be achieved along with all our high-level requirements. There is not a marketed pedal with the same unique features as ours. We even managed to increase the ease of use from our original design and expand on some of our functionality. We had multiple modes of operation for continuous and discrete outputs. With the switches used in place of rotary potentiometers the pedal will be operable while playing guitar. Our final project can synthesize multiple different harmonies. Additionally, it can select between multiple waveforms while also allowing combination with the guitar signal.

5.2 Uncertainties

Right before our final demo while trying to enclose the final product the Teensy begun to heat up and stopped functioning. We were able to read up to 67 degrees Celsius. While there is no exact internal thermal rating for the Teensy 3.2 our Teensy never was warm to the touch of measured above 40 degrees Celsius until that moment. Considering our design had to include wires from the prototyped circuit board due to not predetermining the connections to Teensy on our final PCB order, touching connections is possible. From research a likely cause of failure is that the 5V pin of the Teensy was mistakenly connected any analog pin of the Teensy. All analog pins of the Teensy are rated for only 3.3v [2]. Another reason to believe this is the Teensy was being moved around to be secured in the enclosure which could of cause wire connections to touch.

5.3 Ethical considerations

As the IEEE code of ethics says we must “hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development” [6]. In order to do this, we must consider the fact that someone can be unaware of the risks of the system and make the audio output louder than safe for people's ears. Anything 80 dB and above can cause hearing damage. To comply with an ethical standard our product will include a warning with the range of dB that can cause hearing damage.

It is also a concern that because we are using circuitry that our design is rain proof. Water can cause short circuiting which is a fire safety hazard. To do this we must make sure that our circuit design is sealed enough to not let any water in. We would also need to ensure that all voltages and currents are appropriately grounded to not make the strings of guitar live wires. This can be extremely dangerous for the player and adhering to design standards will help prevent this.

The risk of electrical shock if mishandled also would be a safety hazard to children. From the AMC ethic code 1.2 to avoid harm our team wants to avoid any possible risk to anyone's safety [3]. Our team plans to put a safety warning on the product to keep away from small children to avoid a hazard like this from occurring.

5.4 Future work

While we our happy with our product there are many ways to make it even better. For starters the physical design could be better by adding engraved labels for each switch and connection. As well as adding a led for signaling whether the power is on. Another way to expand on our project

is to add more of what we already have: harmonies, waveforms, and different effects. The one complication of this is we would need an additional switch for each addition which could overcrowd the board and make it less convenient to use. A solution to this would be using rotary potentiometers but have multiple with all the same selections so you can still select multiple waveforms and harmonies. The tradeoff would be making it hard to use with your foot while playing but vastly increased functionality would be worth it.
Another improvement would be adding a rotary potentiometer to adjust gain across the input channels. This would make is so that the combination of the guitar signal and the additional signals would not need to have equal contribution to the output sound. This would allow the guitarist to customize their sound to a higher degree.

References

[1]"Annual Reports | Illini Success - Illinois." *University of Illinois at Urbana Champaign.* 2021, Available at: https://illinisuccess.illinois.edu/annual-reports/

[2] "Audio System Design Tool for Teensy Audio Library" 2021, Available at: https://www.pjrc.com/Teensy/gui/

[3] "ACM Code of Ethics and Professional Conduct." *The Association for Computing Machinery*, 2021, Available at: www.acm.org/code-of-ethics

[4] A. Swanson, "Latency and Its Effect on Performers." Church Production Magazine, 19 June 2017, Available at: https://www.churchproduction.com/education/latency-and-its-affect-on-performers/

[5] B. Neunber, "Pedal Power Basics." *Neunaber Audio*, 2021, Available at: https://neunaber.net/blogs/neunaber-audio-blog/13849473-pedal-power-basics

[6] "IEEE Code of Ethics." 2021, https://www.ieee.org/about/corporate/governance/p7-8.html

[7] Nicholas, "Doubling Guitar with Synthesizers - zZounds Music Blog." *zZounds Music Blog*, 2018, Available at: https://blog.zzounds.com/2017/12/20/doubling-guitar-tracks-with-synthesizers/

[8] "SparkFun Electronics." 30 Sept. 2021, www.sparkfun.com/products

[9] Tom, "Electric Guitar Output Voltage Levels." 2021, Available at: http://tomsguitarprojects.blogspot.com/2014/12/electric-guitar-output-voltage-levels.html

[10] "Op Amp Summing Amplifier: Virtual Earth Mixer » Electronics Notes." 2021, www.electronics-notes.com/articles/analogue_circuits/operational-amplifier-op-amp/virtual-earth-mixer-summing-amplifier.php

[11] Byron, "Proto Pedal Example: Programmable Digital Pedal" 2021, Available at: https://learn.sparkfun.com/tutorials/proto-pedal-example-programmable-digital-pedal/all#assembly-part-2-Teensy-and-controls

[12] Mottola, "Table of Musical Notes and Their Frequencies and Wavelengths" 2020, Available at: https://www.liutaiomottola.com/formulae/freqtab.htm

[13] “Tom’s Guitar Projects”, "Electric Guitar Output Voltage Levels" 2014, Available at: https://tomsguitarprojects.blogspot.com/2014/12/electric-guitar-output-voltage-levels.html

[14] D. Herres, "Waveforms in oscilloscopes and elsewhere" 2017, Available at: https://www.testandmeasurementtips.com/waveforms-oscilloscopes-elsewhere/

Appendix A Requirement and Verification Table

Table 3 Hardware System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)	Explanation
Input buffer subsystem should be able to divide the peak-to-peak voltage of an input signal by approximately 3. It should be able to do so for signals between 2V and 15V peak-to-peak voltage.	1. Used oscilloscope to measure input and output signals of subsystem 2. Subtracted highest peak and lowest peak of both waveforms 3. Do so with waveform generator for 2v and 15v peak to peak	Y	
The output of the input buffers impedance is greater than 100kΩ to prevent signal attenuation	1. Used multimeter to measure the Current and Voltage at the output of the input buffer. 2. Unconnected the input buffer from Teensy and connected multimeter across Teensy and the input buffer. 3. Calculated Z=V/I.	Y	
Output buffer should be able to multiply the peak-to-peak voltage headroom of an input signal by approximately 3. It should be able to do so for signals between 0.5V and 5V peak-to-peak voltage.	1. Used oscilloscope to measure input and output signals of subsystem 2. Subtracted highest peak and lowest peak of both waveforms 3. Do so with waveform generator for 0.5v and 5v peak to peak	Y	
The output buffer impedance is less than 100kOhms to prevent signal attenuation	1. Used Multimeter to measure voltage. 2. Unconnected the output buffer from Teensy and connected multimeter across Teensy and the output buffer. 3. Calculated Z=V/I.	Y/N	Verified but mistake in Design document said greater than instead of less than.
The output from the DAC and the raw guitar signal should be balanced in rms Voltage.	1. Used oscilloscope to measure the waveform of the guitar signal and the waveform from the output of DAC	N	Original Design changed the output of DAC is now already

	2. Calculated Voltage root mean squared using 0.7 times peak voltage.		combined with the guitar signal. Balancing is guaranteed by software.
Be powered by an external 9v power supply rated under 500mA.	1. Use 9v power supply rated under 500mA and test other verifications of the power supply with it.	Y	
The voltage conversion from 9V to 5V should be accurate within ±5%.	1. Use multimeter to measure voltage from input and output of LM7805. 2. Calculate percent error of output voltage from 5v 3. Calculate percent difference error from 9v-5v and the difference between the measured values.	Y	
The temperature of the LM7805 should not exceed 40 degrees C, to ensure that none of the other parts are negatively affected and that the pedal itself doesn't become too hot.	1. Use temperature sensor that attaches to a multimeter to measure the temperature of the pins of the LM7805	Y	

Table 4 ADC/DAC System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)	Explanation
Output is equal to or less than $1V_{rms}$. This is because $1V_{rms}$ is a little more than the highest guitar output voltages.	1. Use the serial monitor and rms voltage function in the Teensy Audio Library to calculate the rms voltage of the output. It should be well under 1V, so that it is not very difficult to balance with the original guitar signal.	Y	
DAC should be able to convert in the frequent range of 50-3000Hz with maximum ±1% signal distortion. This is a tighter boundary, as these are the frequencies that the user will actually	1. An oscilloscope can be used to measure the signal frequency after the output buffer, and the internal note2freq serial monitor can be used to measure the	Y	

hear and they need to be a lot more accurate than the ones required for note computation.	signal frequency from before D-A conversion. They should be within ±1% of each other.		
ADC input is equal to or less than $1v_{rms}$. This is because 1Vrms is a little more than the highest guitar output voltages [12].	1. Use the serial monitor and rms voltage function in the Teensy Audio Library to calculate the rms voltage of the input. It should be less than 1v, since 1v is only rarely reached even with high-gain guitar pickups and strong pick strokes.	Y	
The digital signal is communicated to the microcontroller.	1. Use the serial monitor and rms voltage function in the Teensy Audio Library to calculate the rms voltage of the input. 2. It should be less than 1V, since 1V is only rarely reached even with high-gain guitar pickups and strong pick strokes.	Y	
ADC should be able to convert in the frequent range of 50-1500Hz with maximum ±5% signal distortion. This is because even with ±5% signal distortion, the fundamental frequency will be close enough to a 'correct' note that it can be resolved to it.	1. An oscilloscope can be used to measure the signal frequency after the input buffer, and the internal note2freq serial monitor can be used to measure the signal frequency from the ADC. They should be within ±5% of each other.	Y	

Table 5 DSP Subsystem Requirements and Verifications

Requirement	Verification	Verification status (Y or N)	Explanation
Should be able to resolve a frequency to the nearest 'correct' note within 5 milliseconds.	1. This can be verified by attaching an oscilloscope to the input signal from the guitar as well as the output from the frequency analyzer, and from the plot of the	Y	

	waveforms calculating the time difference between the starts of the two signals		
Tie break to the higher note.	1. This can be ensured by using the waveform generator to input a frequency exactly halfway between two notes, and ensuring that the output note is the higher one	Y	
Latency should be under 5 milliseconds.	1. This can be verified by attaching an oscilloscope to the input signal from the guitar as well as the output from the frequency analyzer, and from the plot of the waveforms calculating the time difference between the starts of the two signals.	Y	
Should be able to calculate at least the following harmonies: unison, fifth, major third, minor third, major 7th, minor 7th.	1. The outputted notes can be manually checked against chord charts to ensure that the correct harmonies are being generated.	Y	
Should allow the guitarist to quickly select the desired harmony setting	1. This can be done by checking if a guitarist can use only one hand to change the settings, and if it can be done within ~2 seconds	Y	
The MIDI notes should be able to drive any MIDI-controlled device and should externally appear as a standard MIDI controller.	1. To ensure that the output is MIDI standard compatible, one can connect a laptop and use multiple digital instruments in a DAW, and compare the generated notes with the expected notes using a software keyboard.	N	Unfortunately, our expensive MIDI connectors got lost in shipping so we were unable to actually test the MIDI port

Appendix B Schedule

Table 6 Schedule

Week	Danielle	Madhav
Aug 23rd-Aug 29th	-Commented on an Initial post	-Made idea post
Aug 30th-Sept 5th	-Lab Safety Training -CAD assignment	-Lab Safety Training -CAD assignment
Sept 6th-Sept 12th	-Project approval submitted	Project approval submitted
Sept 13th-Sept 18th	-Soldering assignment -Did project proposal -Researched guitar terms and made study guide	-Soldering assignment -Did project proposal -Helped Danielle get acquainted with guitar terminology
Sept 20th-Sept 26th	-Started Design Document	-Started Design Document
Sept 27th- Oct 3rd	-Research the PCB layout -Practice speaking for Design review -order Teensy Audio Board	-Research most suitable microcontroller and parts -Practice speaking for Design review -order Teensy audio board
Oct 4th-10th	-Research the PCB layout about Teensy microcontroller/audio board -Had Design review -Eagle the PCB layout	-Researched frequency estimation algorithms -Had Design review -Eagle the PCB layout
Oct 11th-17th	-Redid entire PCB on Eagle	-Researched audio libraries
Oct 18th-24th	-Met in lab, ordered all missing parts	- Ordered missing parts, started testing Teensy with audio library
Oct 25th- 31st	-	
Nov 1st-7th	-Built the power, input buffer, output buffer, and combinational buffer on Breadboard -Verified power, input buffer, output buffer on breadboard	- Programmed Teensy with barebones guitar doubling code - Debugged Teensy port connections - Got good results for baseline functionality on breadboard
Nov 8th-14th	-Redid the Schematic and PCB design on Eagle	-Ordered new PCB with Danielle -Wrote code for advanced functionality
Nov 15th-21th	-Tried to get good sound from combinational logic on board	-Ordered another PCB just to be sure

	-Soldered a backup PCB	-Had a smaller footprint, ended up using it
Nov 22nd- 28th	-Used copy of PCB to debug the power -Debugged output buffer -Worked on getting signals to combine	-Used new PCB to get I/O buffers, Teensy with Audio Shield working -Got great results with Audio Shield, But better cost effectiveness with DAC chip -Prepared chassis with switches and digital connections
Nov 29th- Dec 5th	-Work on mock demo -insulate project -Collect remaining verifications for my part	Work on mock demo -insulate project and put in chassis -Collect remaining verifications for my part
Dec 6th- 13th	-Continue to work on Presentation -Present -Do my parts of final report including conclusion, block diagram, cost, hardware system	-Final Presentation -Final Paper

Appendix C Complete DSP Code in C/Arduino

```
#define GUITAR_NOTE_COUNT 49
#define FREQUENCY_MULTIPLIER 0.9666

#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SerialFlash.h>


AudioInputAnalog         adc1;           //xy=291,153
AudioSynthWaveform       waveform4;      //xy=298,573
AudioSynthWaveform       waveform3;      //xy=304,524
AudioSynthWaveform       waveform2;      //xy=305,470
AudioSynthWaveform       waveform1;      //xy=306,422
AudioAnalyzePeak         peak1;          //xy=538,517
AudioMixer4              mixer1;         //xy=579,381
AudioAnalyzeNoteFrequency notefreq1;      //xy=627,155
AudioMixer4              mixer2;         //xy=774,253
AudioOutputPT8211        pt8211_1;       //xy=955,293
AudioConnection          patchCord1(adc1, notefreq1);
AudioConnection          patchCord2(adc1, 0, mixer1, 0);
AudioConnection          patchCord3(adc1, peak1);
AudioConnection          patchCord4(waveform4, 0, mixer2, 1);
AudioConnection          patchCord5(waveform3, 0, mixer1, 3);
AudioConnection          patchCord6(waveform2, 0, mixer1, 2);
AudioConnection          patchCord7(waveform1, 0, mixer1, 1);
AudioConnection          patchCord8(mixer1, 0, mixer2, 0);
AudioConnection          patchCord9(mixer2, 0, pt8211_1, 0);

void setup() {
  // put your setup code here, to run once:
  AudioMemory(100);
  notefreq1.begin(0.15);

  waveform1.begin(WAVEFORM_SINE);
  waveform1.amplitude(0.1);
  waveform1.frequency(50);
  waveform1.pulseWidth(0.15);

  waveform2.begin(WAVEFORM_SINE);
  waveform2.amplitude(0.1);
  waveform2.frequency(50);
  waveform2.pulseWidth(0.15);
```

```
  waveform3.begin(WAVEFORM_SINE);
  waveform3.amplitude(0.1);
  waveform3.frequency(50);
  waveform3.pulseWidth(0.15);


  waveform4.begin(WAVEFORM_SINE);
  waveform4.amplitude(0.1);
  waveform4.frequency(50);
  waveform4.pulseWidth(0.15);

  mixer1.gain(0, 5);
  pinMode(2, INPUT);
  pinMode(4, INPUT);
  pinMode(6, INPUT);
  pinMode(8, INPUT);
  pinMode(10, INPUT);


}

void loop() {
  // put your main code here, to run repeatedly:
float guitar_notes[GUITAR_NOTE_COUNT] = {82.41, 87.31, 92.50, 98.00, 103.8, 110.0,
116.5, 123.5,
 130.8, 138.6, 146.8, 155.6, 164.8, 174.6, 185.0, 196.0, 207.7, 220.0, 233.1, 246.9,
  261.6, 277.2, 293.7, 311.1, 329.6, 349.2, 370.0, 392.0, 415.3, 440.0, 466.2, 493.9,
 523.3, 554.4, 587.3, 622.3, 659.3, 698.5, 740.0, 784.0, 830.6, 880.0, 932.3, 987.8,
  1047,  1109,  1175,  1245 , 1319};

   int wave_style_continuous = digitalRead(2);
   int major_minor = digitalRead(4);
   int octave_on = digitalRead(6);
   int third_on = digitalRead(8);
   int fifth_on = digitalRead(10);

//   if(wave_style){}


  if (peak1.available())
{
    float peakv = peak1.read();

    waveform1.amplitude(sqrt(10000*peakv));
    waveform2.amplitude(sqrt(10000*peakv));
    waveform3.amplitude(sqrt(10000*peakv));
```

```
      if (notefreq1.available())
{


        float target_freq = notefreq1.read();

        float prob = notefreq1.probability();

      float note = bin_search(guitar_notes, 0, 48, target_freq *
FREQUENCY_MULTIPLIER);

        if (!wave_style_continuous){
          if (target_freq < 80){target_freq = 82;}
          if (target_freq > 1400){target_freq = 1350;}
          }
        else{note = target_freq;}



        if(octave_on){waveform1.frequency(note * 0.5);}
        else{waveform1.frequency(0);}

        if(fifth_on){waveform2.frequency(note * 1.49831);}
        else{waveform2.frequency(0);}

        if(third_on){
          if(!major_minor){
            waveform3.frequency(note * 1.18921);
            waveform4.frequency(note * 1.782);
            } // minor
          else{
            waveform3.frequency(note * 1.25992);
            waveform4.frequency(note * 1.888);
            } //major
          }
         else{waveform3.frequency(0);}


        Serial.printf("Frequency:%3.2f | Note:%3.2f | Probability: %.2f | PeakIn:
%.5f\n",target_freq, note, prob, peakv);
     }



   }
```

```
}


float bin_search(float *arr, int start_idx, int end_idx, float search_val) {

   if( start_idx == end_idx )
      return arr[start_idx] <= search_val ? arr[start_idx] : arr[start_idx];

   int mid_idx = start_idx + (end_idx - start_idx) / 2;

   if( search_val < arr[mid_idx] )
      return bin_search( arr, start_idx, mid_idx, search_val );

   float ret = bin_search( arr, mid_idx+1, end_idx, search_val );
   return ret == 0 ? mid_idx : ret;
}
```