Capstone Design Course Report:

Synthesizer

Prepared for: Prof. Mark Bauer

Prepared by: Daniel Andrews, Juhyung Park, Paul Circo

December 10, 2021

Department of Electrical Engineering, College of Engineering

University of Nebraska-Lincoln

ABSTRACT

Design Team: G157
Team Member: Daniel Andrews, Juhyung Park, Paul Circo
Design Advisor: Prof. Mark Bauer
Course: ECEN 495

This report is written for the course ECEN 495 in Fall 2021 at the University of Nebraska-Lin coln. In the class, Daniel Andrews, Juhyung Park, and Paul Circo were in group G157. The gr oup's project is to create a synthesizer with a microcontroller. We have decided to make a por table synthesizer. A push button interface will provide the user with controls for different opti ons. The user will be able to choose between waveforms/tones, which will then drive a stand ard 3.5 mm headphone jack.

TABLE OF CONTENTS

Introduction

This report presents the original concepts behind the project and delves into the phases of work. This team was created and started working on this project in the Spring 2021 semester.

Here is an overview of this design report:

1. Abstract
2. Introduction
3. Body
4. Parts List
5. Intellectual Property
6. References
7. Appendix

A growing number of consumer electronics are being designed in smaller form factors for both ease of use and portability. One application in particular stood out to us. Many new solid-state guitar amplifiers allow a guitarist a portable system with the ability to change amp settings without having a cumbersome control interface. These minimalist devices are very practical for traveling, or other applications where small forms are beneficial. To those of us in the group, it appeared that there was a disparity between guitar products and keyboard/synth products in this area. This was the fundamental thought process behind our project. How could a synthesizer be downsized in a more portable, travel-conscious way? Our synthesizer could read MIDI inputs to allow for higher levels of control compatibility, without the larger footprint. We could have a user interface that allowed for a few more advanced features, and a standard 3.5mm audio output jack for headphone use.

With this in mind, we have tried to ensure that our synthesizer satisfies our intended purpose. We are able to make functions and convert them into the sound that we want. User inputs on the button array will generate a corresponding waveform which will then run through an equalizer circuit and distortion circuit. We can switch between square, sawtooth, and triangle waves, and can run through ascending or descending frequencies.

This paper will present the outline of the synthesizer, the sources of our design, the principle of the synthesizer, and the results.

Body

1. Microcontroller

The microcontroller used for this project is the STM32F103C8, with development and testing being done on the STM32F103C8T6 Development Board, otherwise known as the "Bluepill." To help with the initialization of our programming environment to set up the Bluepill, STM32CubeMX is used to generate the initial code. From there, development is done in the Keil uVision5 IDE using the C programming language.

2. Oscillator

The waveform oscillators are developed through code within the Bluepill microcontroller. Currently, code to generate a single sawtooth, square, or triangle wave has been implemented with controllable amplitudes and frequencies. The square and saw wave functions have a void return type and take in a uint8_t preload value and a uint8_t value for amplitude as arguments. The triangle wave function is a bool return type with arguments of a uint8_t preload value, uint8_t amplitude value, and a boolean value that determines if the wave is on the rising or falling cycle. In the main function, once a frequency, amplitude, and waveform are selected, the prescale value is calculated. With the 72 MHz clock frequency, a predetermined preload value, and the newly calculated prescale value, Timer1 is initialized. If any of these values change, Timer1 will be re-initialized. Now, as the code is running, it is waiting for the gpio input from the play button to enter a while(button is pressed) loop. Within the while loop, the appropriate oscillator function is called and run. An integer variable is constantly updated with the current timer value, and once this variable is equal to the prescale value, the code will update the output data register associated with an 8-bit GPIO register accordingly. For the square wave function, the ODR is simply toggled between 0 and whatever the amplitude value

was selected to be. For the sawtooth function, the ODR begins at 0 and is ticked up by a val ue of 1 every time the current timer value equals the prescale value. The ODR is then reset to 0 whenever it reaches the amplitude value. The triangle wave function behaves very similarl y to the sawtooth wave function, except once the ODR reaches the amplitude value, it begins to tick back down until it reaches 0, which then begins to tick back up again. Whenever the tr iangle wave function reaches its peak, it will toggle the boolean value that was taken in so tha t the direction is switched the next time it's incremented. The new boolean value is then retur ned to itself. Oscilloscope images of each oscillator output can be viewed in Figures 1, 2, and 3.

3. Digital to Analog Converter

To produce an analog audio signal from our microcontroller, a digital to analog conve rsion system is necessary. The digital waveform output will be written as an 8-bit signal, whi ch is sent to an 8-bit GPIO register. This GPIO register will act as the digital inputs on the D AC0808LCN D/A Converter integrated circuit chip. The output of the DAC0808LCN will th en be fed into an op-amp to amplify the signal. Currently, the op-amp being used is a UA741 CP. The circuit used in this process can be seen in Figure 4.

4. Equalizer

Before we could begin to design the equalizer, we first tried to come up with a range o f acceptable frequency boundaries. To start with, human hearing is generally capable of heari ng in the 20 hertz to 20,000 hertz. From here, we obviously have to pick out frequency range s. From our research, it appeared that 100-200 hertz is typically used for the bass band. Also, we found that most of the midband frequencies were around 1kHz. Determining a high freq uency was more of a matter of preference. While we might be able to hear up to 20kHz, we d

efinitely thought that frequencies were not musical before that point, more or less after 2kHz. Above that threshold, the notes just seemed irritating, even painful.

We then began prototyping the board out. For bass, we assumed 100 Hz to be a reason able cutoff frequency since the filtering will not instantaneously cut the signal. After doing s ome breadboard testing, we reasoned that maybe the bass cutoff region should include some f requencies that were a little higher, as A/B comparing input and output with an oscilloscope s howed less dramatic effect than we expected. We found some online resources that seemed t o agree with that conclusion[1], and recommended 200 hertz for a cutoff frequency. Otherwi se, we stuck with our 1kHz and 2kHz values.

In terms of function, the input signal first runs through some capacitors, and then into an op-amp. From there, the signal is split into our three bands for our desired frequencies. T he wipers of each band's potentiometer lead to the input of a final op-amp to allow cut and bo ost ability. The band signals then merge back tack together and continue through the circuit t oward the distortion. The design for our equalizer can be seen in Figure 5. At this point, the signal is split into two, one path for distortion, and one clean pathway.

5. Distortion

We thought it would be an important feature to allow the end user to add distortion to the signal if they so desire. Typically, the main means for distortion are either a "soft" diode clipping circuit or a "hard" diode clipping circuit. Soft diode clipping fundamentally feature s a set of antiparallel diodes in the feedback loop of an op-amp. Sometimes other elements ar e added in parallel to the diodes, such as capacitors and resistors (to control the output level). In contrast, hard diode clipping typically features antiparallel diodes that run from an output t

o ground. Again, sometimes other elements (like resistors) are added to control the level of d istortion applied.

To test which distortion method we wanted to use, we ran signals through our equaliz er and then compared hard clipping, soft clipping, and combining the two methods. Between the three methods, we found that combining these topologies produced the best sounding dist ortion when applied to our synthesized waveforms. The design for our distortion can be seen i n Figure 6.

6. Digital Potentiometers

The MCP4231 and the DS1881 were the digital potentiometers that we used on our ci rcuit boards to control different aspects of the synthesizer. The MCP4231 is a linearly tapere d potentiometer that can be controlled through SPI. This potentiometer was used to blend bet ween the clean signal and the distortion signal. This allows the user to control how much dist ortion they want in the signal path. The DS1881 is a logarithmically tapered audio potentiom eter that can be controlled through I2C. Each DS1881 contains two potentiometers, so by usi ng 2 of these devices, we can control our three EQ bands, as well as our output volume. You can see the two DS1881s in Figure 7. The MCP4231 was shown in Figure 6.

7. Output

After the signal has passed through the equalizer (and any distortion in which the user has enabled), the signal will come to one final op-amp. This final op-amp is used so that the output is capable of driving either headphones or a speaker line out. On both the breadboarde d prototype and the EQ PCB, we used a breakout board for an audio jack. This output jack is of the TRRS 3.5mm type. The output is mono sound, meaning the left ear and right ear signa l are the same. Typical mono sound would not require a TRRS jack, but using one is best prac

tice, as this would allow for maximum flexibility. The layout of TRRS jacks can be seen in F igure 8. TRRS jacks allow the possibility of left/right ear specific output, as well as micropho ne input or control. In this project, the left and right output were tied together. We did not see the increase in complexity as a worthwhile avenue to pursue.

8. User Interface

A simplistic approach was taken when designing the user interface. Four push button s were implemented to act as active high GPIO inputs. The first button, when pressed, would call and run the oscillator function until it was released. The second button was used to toggl e through which oscillator was currently selected. The third button would lower the note freq uency by an arbitrary value, while the fourth button would raise it.

9. Bluetooth Connection

In the original planning stages of this project, we intended to do most of the controllin g of the synthesizer via Bluetooth connection. The intent was to communicate with an iOS a pp (as is explained in the next section). For a Bluetooth connection, we got the H3 Dual Bluet ooth Module. In our test setup, we were able to have a connection to an android device and r ead the serial data using a usb to serial converter on a PC. We were also able to connect to th e H3 module with a PC and send data back and forth and be able to properly read the data. T his was an exciting development for us, because we lacked Bluetooth experience. Unfortunat ely, this is where we encountered our issue with Bluetooth. While the H3 was adequate to con nect to PCs and Android devices, the H3 was not discoverable on any iOS devices. None of our group members had experience with this issue, and we concluded from some research tha t this was likely due to differing security requirements by iOS and Android. This was an issu e, because the only experience the group had in app development was *specifically* for iOS. A

t this point, the group evaluated the importance of the Bluetooth component of the project to determine whether or not it was worth pursuing. We ultimately decided that the best course of action was to eliminate Bluetooth communication from the project.

10. Control App

The original plan was for the parameters of the synthesizer to be controlled by an iOS app. The app would connect to the synthesizer through Bluetooth and would provide a GUI t o control the parameters of the oscillator. The idea was to switch on a variety of effects like e nvelope, flanger, delay, equalizer, reverb, and distortion. The user would also be able to chan ge the volume level. The app would be a tabbed app, to make it easy to navigate between set s of controls, similar to how you would use tabs in a web browser. Each tab would have its o wn GUI to control a single component of the synthesizer. The data from the digital sliders an d buttons within the app would be transferred through Bluetooth to the microcontroller. After it was realized that Bluetooth connections to Apple products was unlikely, the idea for a cont rol app was scrapped.

11. MIDI

Originally, the group planned on using MIDI over USB for the input to the synthesize r. After considering the implications of USB vs standard MIDI cable, we opted to use a cable d input. The idea was for an external MIDI keyboard to be connected with a MIDI cable, and then the codes being read into the microcontroller through UART. The MIDI codes will tell t he synthesizer what notes to trigger, their velocity, and their timing. The group then procured a MIDI keyboard to use for the project that had a midi cable jack. At this point, we began res earching for the proper circuit to read MIDI signals. We found two different schematics that

appeared to work in theory. Both schematics used an optocoupler to isolate the input and out put. This makes quite a bit of sense, as a user is not going to want to worry about a bad cable destroying their MIDI instrument (especially with the cost involved in a good MIDI device). The group purchased a small quantity of 4N25 optocouplers, MIDI connectors, cables, etc. a nd began assembling the input circuit.

Once the circuit was constructed, we connected the keyboard and input circuit. We th en connected a logic analyzer with a MIDI decoding mode onto the circuit. Despite our keyp resses on the keyboard, we could not get any output on the output side of the optocoupler. W e checked to make sure that we were in the correct operating range of the optocoupler, and di dn't find anything out of spec. Next, we tried reading on the input side of the optocoupler, to see if we were even getting any signal through the cable. What we observed was that we wer e receiving a portion of orphaned data followed by a system initialization message. Despite g etting these notes, we still were unable to see the output of any notes whatsoever. We checke d our keyboard's settings and user manual to see if we could find anything that would help. We checked all of the output channel settings (since MIDI allows the ability to have multiple channels to be controlled by one device, as well as a "Global" channel) and could not determi ne the cause of the problem. We ensured the keyboard wasn't on "mute" mode, we checked t he output channel levels, and we checked to make sure that the keyboard was on "cable" mod e and not on USB mode. After checking these things on our keyboard (plus more), and re-rea ding the user manual, we began to wonder if we didn't have an issue with the keyboard. We h ooked an oscilloscope up on the input side of the optocoupler to see what the waves looked li ke. The first thing that we noticed was that the signal was horrifically noisy. We changed the cable out, to be safe, but there was still a profound amount of noise on the signal. We also n

oticed that the output voltage of the keyboard was approximately 1.6 Volts, which seemed rat her out of spec to the expected 5 Volts.

At this point, we decided we would try again with the second style of MIDI input circ uit using a different optocoupler, a 6N138. We decided to switch because this second style w as found to be the alleged standard (which was confirmed after finding the official MIDI spec ifications). We thought this would remedy any issues if the first schematic was improperly d esigned. The proper input diagram is displayed in Figure 9.

At this point, we double checked the keyboard with the proper schematic and got the e xact same results. We still had no signal pass through the optocoupler, and only the orphaned data and initialization codes on the input side of the optocoupler. Our next step was to find a nother keyboard to see if we could get any output with a different input device. We double c hecked our circuit with a far newer, more expensive MIDI keyboard and fared no better. On t his keyboard we no longer received the orphaned data, but received identical system initializa tion messages. We were unable to have constant access to this second keyboard, so that mad e any potential fixes difficult. We were left with some puzzling thoughts. If the first keyboar d voltage was that far out of spec, but we received the same data signal with the newer (presu mably in-spec) keyboard, why were they both unable to function properly? We tried to test t his using a dummy square wave signal and also trying to tie the input high and low. Yet we w ere still unable to receive proper input. The amount of time spent on this issue was rather lar ge, and there were other things in the project that needed done, so unfortunately we were not able to solve this. This problem was an infuriating one, because there seemed to be an extensi ve amount of knowledge on this area available, but nothing that we researched pointed to a sp ecific problem. This was an area that seemed straightforward, and unfortunately was not. At t his point, we decided it was time to plan for an alternative control method.

12. PCB Design

Printed circuit boards were designed using the KiCad software. In order to divide the work up, two different circuit boards were designed to interface with each other. The first bo ard design contained the microcontroller, connection pins for various communication periphe rals, and the DAC circuitry to output an analog audio signal. The second board contained cir cuitry for the equalizer, distortion, and final audio output to the 3.5 mm audio jack, as well as the main power input with voltage regulators to give the various voltage levels needed throug hout the project. The PCBs were ordered through Oshpark and were then put together by sol dering the various surface mount and through hole circuit components. The designed PCBs c an be viewed in Figures 10, 11.

13. Power Supply

The original idea was to supply the project with two 9 volt batteries for the project to be more portable. After we began testing the circuit boards, it became obvious that using a p ower supply was a far easier method. The shortcomings of capacity with a 9 volt battery was also noted. At this point, we decided it was easier to use two 120VAC to 9VDC converters b ecause they could be isolated from each other and easily connect to the +9V and -9V (and gro und) connections we used from the bench power supply.

14. Physical Case

In order to house the project, a PacifiTec project case was used, which was fitted with a custom 3-D printed cover to allow the push-buttons to be accessed. A hole was drilled into the side of the case to allow for an auxiliary cord to be connected to the 3.5mm audio jack. T

he power lines were fed through a gap in what would have been the battery housing on the ot her end of the case. An image of the final product within the enclosure can be seen in Figure 12.

Parts List

- STM32F103C8
- DAC0808LCN D/A Converter
- UA741CP / LM741 Operational Amplifiers
- DS1881 Logarithmic Digital Potentiometer
- MCP4231 Linear Digital Potentiometer
- AMS1117CD-3.3 Voltage Regulator
- LM137 Voltage Regulator
- LM340T Voltage Regulator
- 4N25/6N138 Opto-isolator
- 1N914 Small Signal Diode
- Sparkfun TRRS 3.5mm Jack Breakout

Appendix

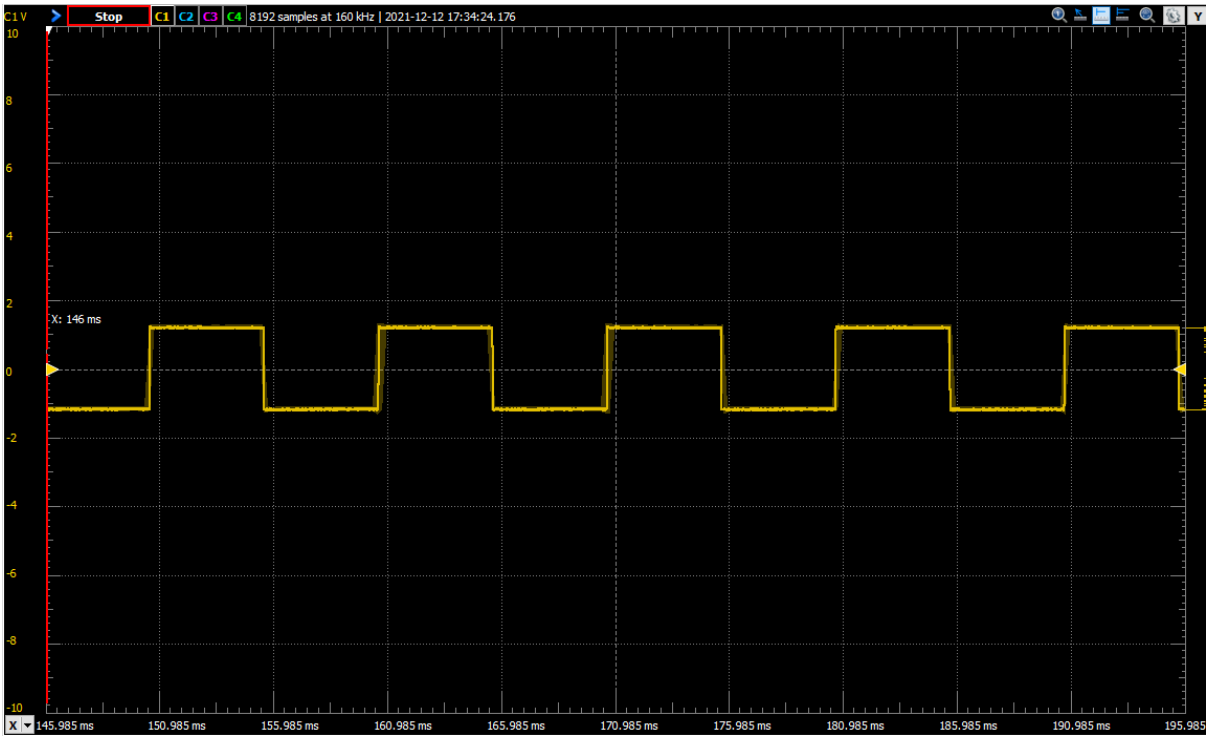

Figure 1 Square wave oscillator output

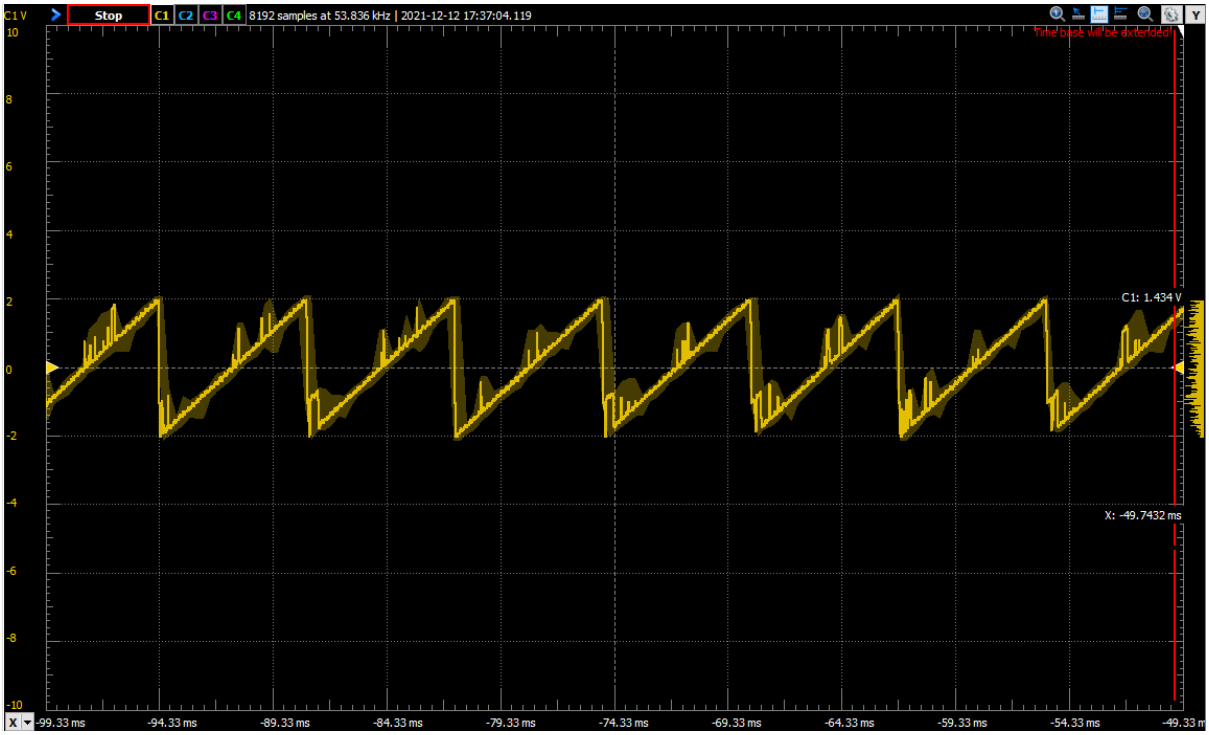

Figure 2 Sawtooth wave oscillator output

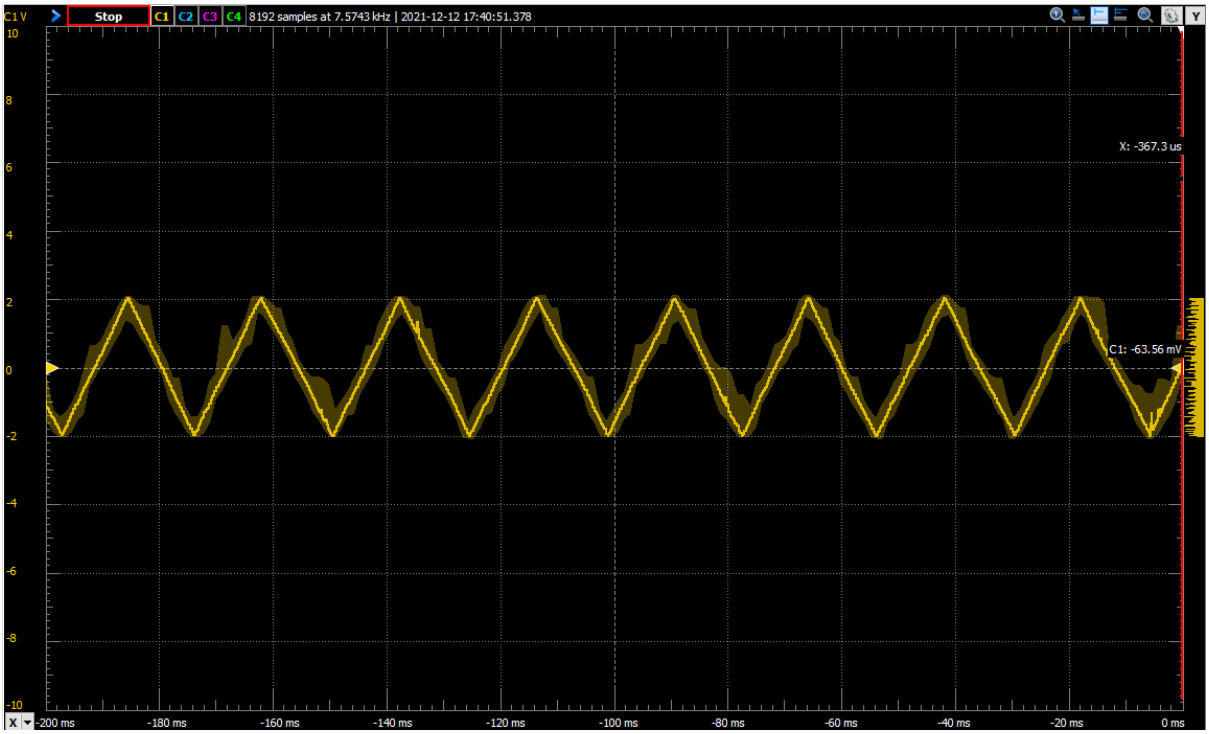

Figure 3 Triangle wave oscillator output

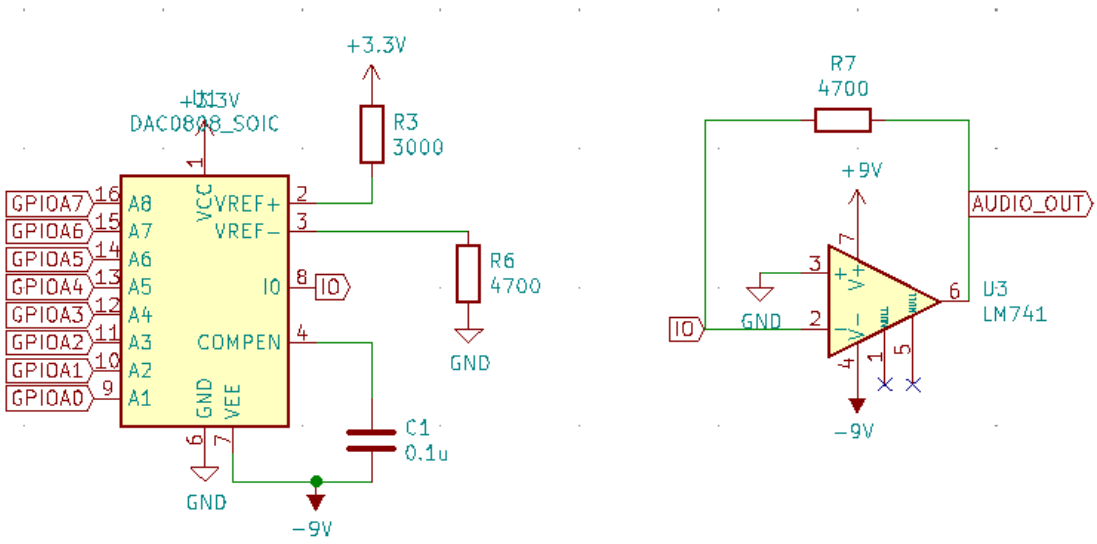

Figure 4 Digital to analog converter circuit

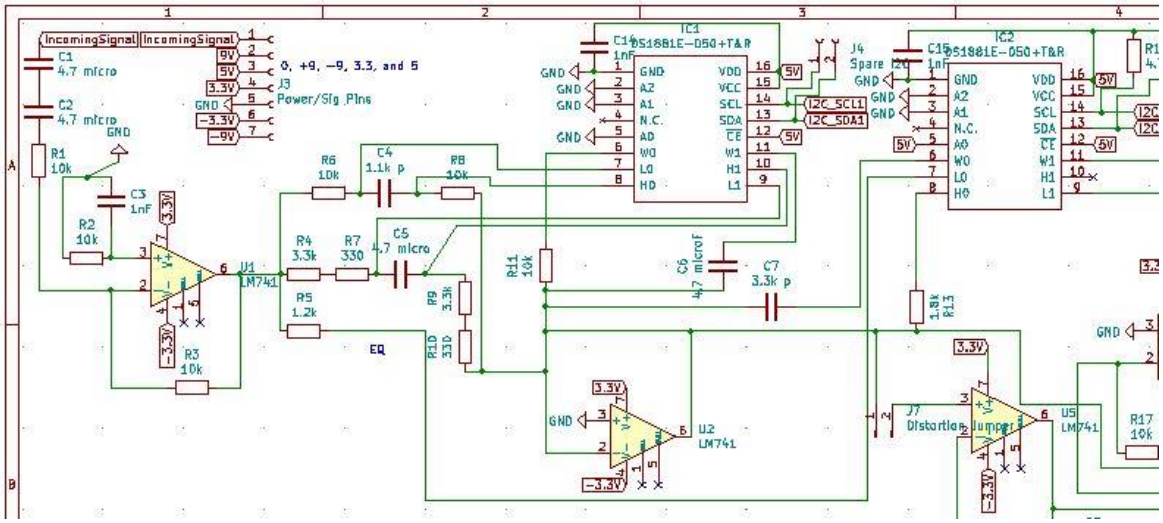

Figure 5 EQ Circuit schematic with the three band paths.

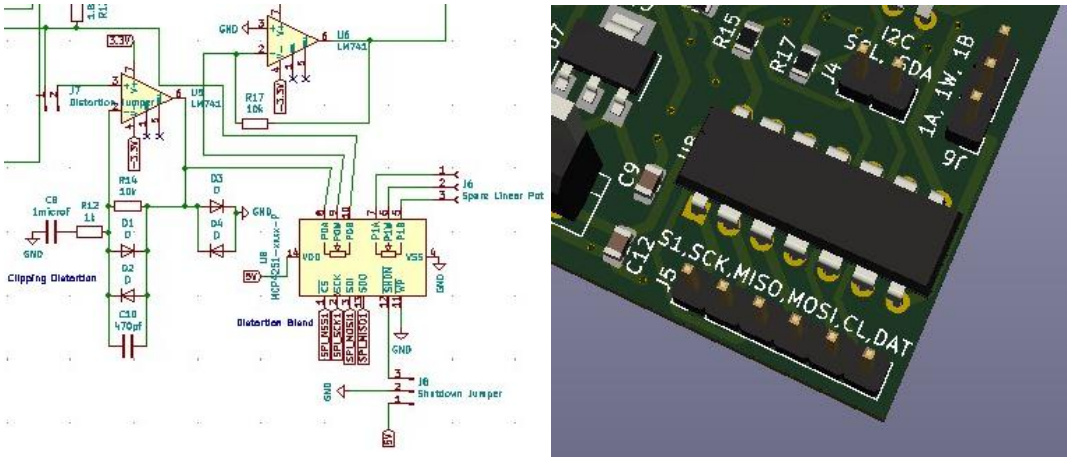

Figure 6 Distortion portion of the circuit with soft and hard clipping diodes. Also shown is t he MCP4231 potentiometer.

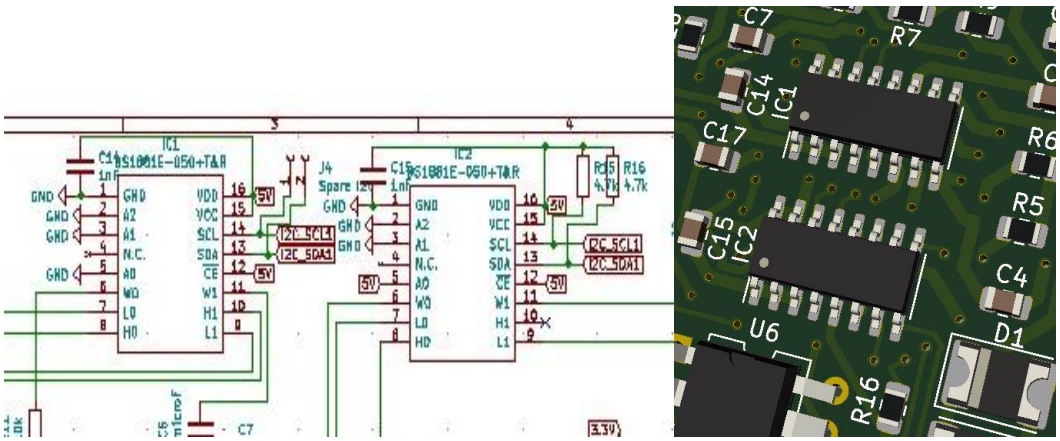

Figure 7 DS1881 logarithmic potentiometers used for controlling the equalizer and the volu me.

Figure 8 TRRS breakout board with TRRS diagram.

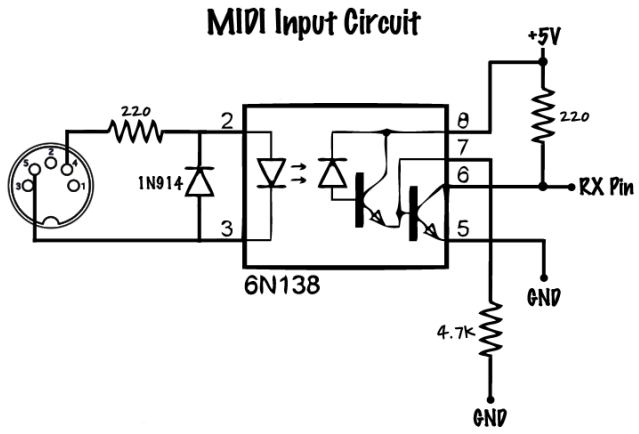

Figure 9 Standard MIDI input circuit with 6N138 optocoupler and 1N914 small signal diode.

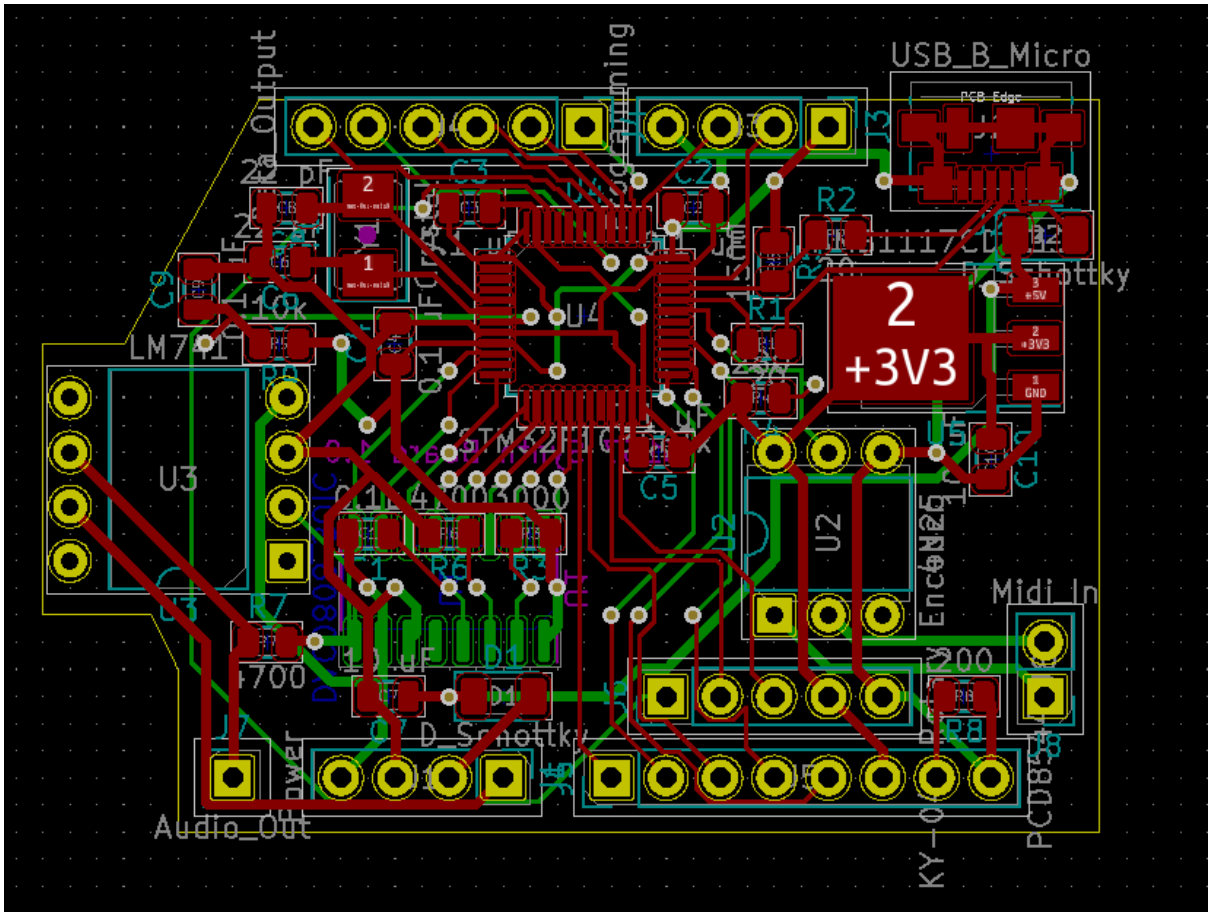

Figure 10 Control PCB layout

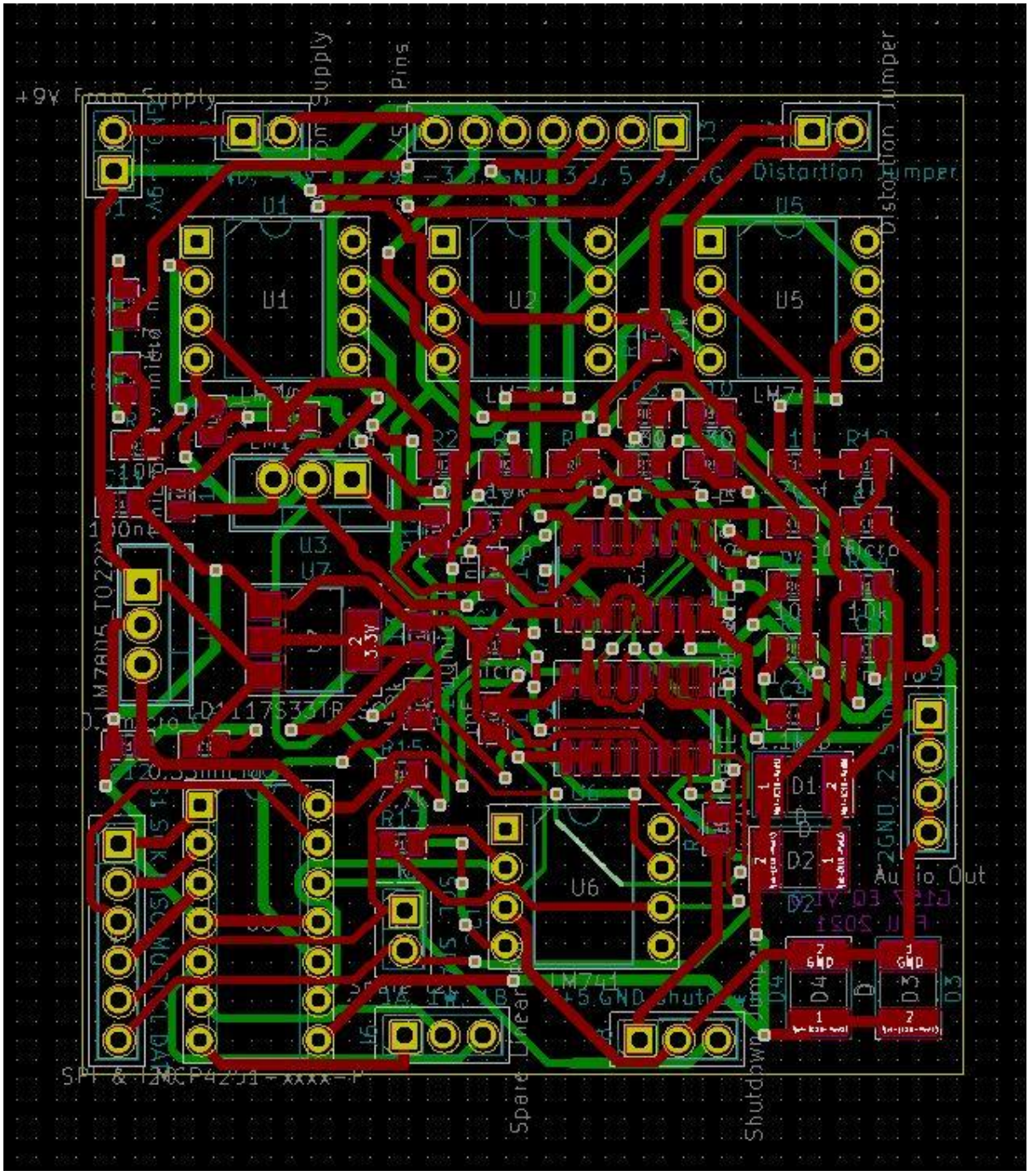

Figure 11 EQ PCB layout

Figure 12 Final enclosure design

References

---

[1] https://www.electroschematics.com/3-band-equalizer/

[2] STM32 reference manual

https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf

[3] STM32 data sheet

https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

[4] DAC0808 D/A Converter Datasheet

https://www.ti.com/lit/ds/symlink/dac0808.pdf

Distortion Design References

[5] https://sessionville.com/articles/how-to-make-your-own-distortion-pedal

[6] https://www.guitarpedalx.com/news/news/a-brief-hobbyist-primer-on-clipping-diodes#:~:text=to%20a%20degree.-,Hard%20Diode%20Clipping,actually%20the%20Klon%20Centaur%20too!

MIDI

[7] https://midi.org/

[8] https://www.notesandvolts.com/